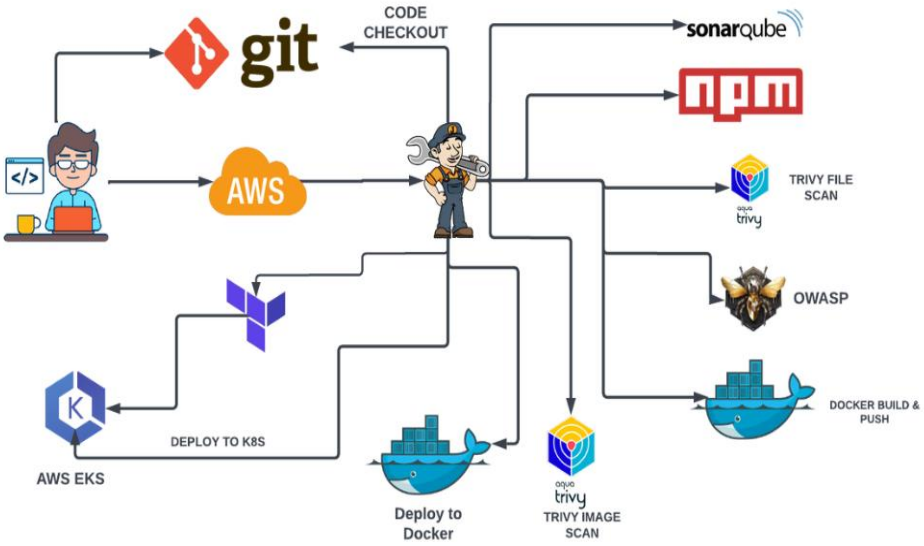
DevSecOps CI/CD Pipeline for an Uber Clone



GIT REPO: https://github.com/vommidapuchinni/uber-clone.git

Overview:

This document provides a detailed overview of the continuous integration and continuous deployment (CI/CD) pipeline implemented for the Uber clone application. It is intended for developers, DevOps engineers, and security analysts involved in maintaining and enhancing the pipeline.

Instructions to do this:

Step1: Launch an ubuntu instance (t2.2xlarge)

Step2: Create IAM role

Step3: Create S3 bucket

Step4: Installation of packages

Step5: Connect to Jenkins and SonarQube servers

Step6: Terraform plugin install and EKS provision

Step7: Plugins installation & setup (Java, Sonar, Nodejs, OWASP, Docker)

Step8: Configure in Global Tool Configuration

Step9: Configure Sonar Server in Manage Jenkins

Step10: Pipeline up to Docker

Step11: Kubernetes Deployment

Step12: Accessing our application

Step13: Destruction

Prerequisites:

1. **Version Control System (VCS)**:
    - Use a VCS like Git (GitHub) to manage your source code. Ensure proper branching strategies are in place (e.g., feature branches, develop, master).
2. **Containerization**:
    - Docker containerization tool to package your application and its dependencies into containers, which ensures consistency across different environments.
3. **Infrastructure as Code (IaC)**:
    - Tools like Terraform to define and manage your infrastructure in a version-controlled and repeatable manner.
4. **CI/CD Tool**:
    - Choose a CI/CD tool like Jenkins, to automate the build, test, and deployment phases.
5. **Security Tools Integration**:
    - Select security tools for static code analysis (e.g., SonarQube), image scanner (Trivy), OWASP dependency check.

Definitions:

1. **GitHub** is a web-based platform for Git repositories, enabling version control and collaborative software development
2. **Git** is a distributed version control system for tracking changes in source code during software development.
3. **AWS (Amazon Web Services)** is a comprehensive cloud computing platform offering on-demand computing resources and services.
4. **EC2 (Elastic Compute Cloud)** is a web service that provides resizable compute capacity in the cloud, designed to make web-scale computing easier for developers.
5. **Terraform** is an open-source infrastructure as code (IaC) tool used for building, changing, and versioning infrastructure safely and efficiently.

6. AWS EKS (Amazon Elastic Kubernetes Service) is a managed Kubernetes service provided by AWS for deploying, managing, and scaling containerized applications using Kubernetes on AWS cloud infrastructure.

7. Docker is a platform that enables developers to package, deploy, and run applications and their dependencies in lightweight, portable containers.

8. **Trivy** is an open-source vulnerability scanner for containers and other artifacts, used to detect security vulnerabilities in software dependencies.
   Trivy supports both file scanning and image scanning to identify security vulnerabilities:
   - ❖ File Scan: Trivy can scan individual files or directories to detect vulnerabilities in software packages and dependencies present within those files.
   - ❖ Image Scan: Trivy can also scan Docker images to identify vulnerabilities in operating system packages, application dependencies, and other components within the container image layers.

9. **OWASP Dependency-Check** is an open-source tool that identifies known vulnerabilities in project dependencies, helping developers manage and mitigate security risks.

10. **npm (Node Package Manager)** is a package manager for JavaScript that helps developers discover, share, and reuse code, managing dependencies and scripts for Node.js projects.

11. **SonarQube** is an open-source platform for continuous inspection of code quality and security, providing detailed reports and analytics to improve software development practices.

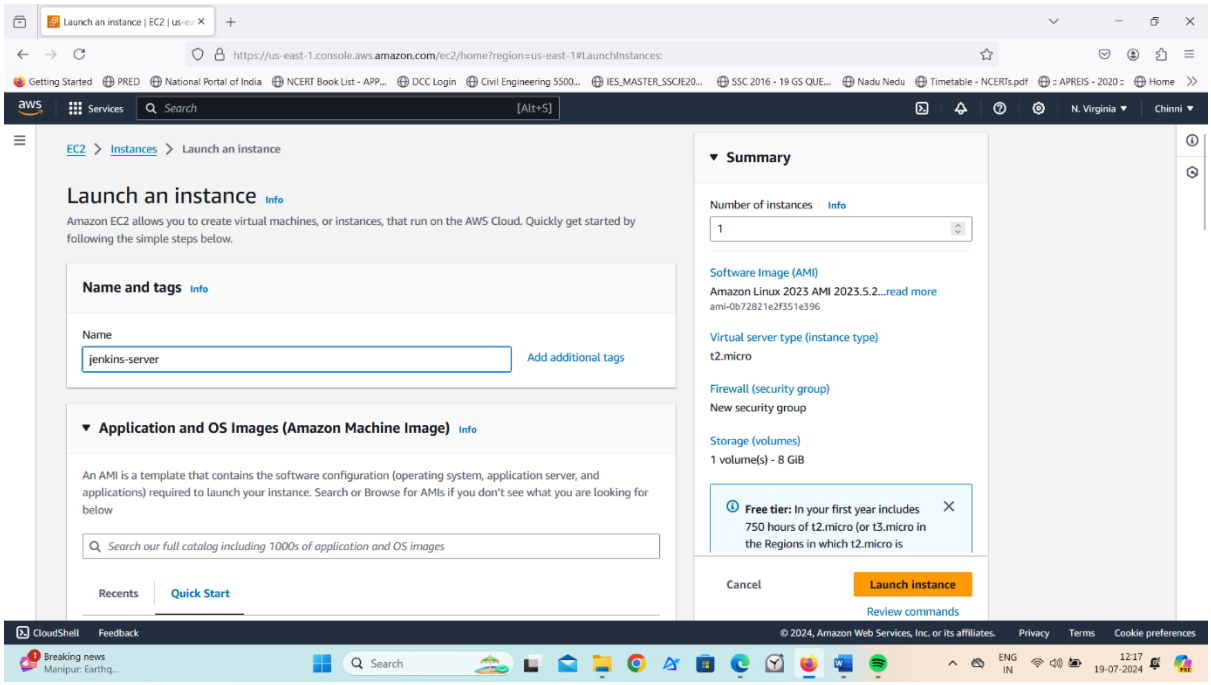
12. **IAM role (Identity and Access Management role)** defines permissions for AWS entities without the need for credentials, enhancing security and access control in cloud environments.

13. **S3 (Simple Storage Service)** is an object storage service offered by AWS, providing scalable storage for data, files, and backups accessible via the internet.

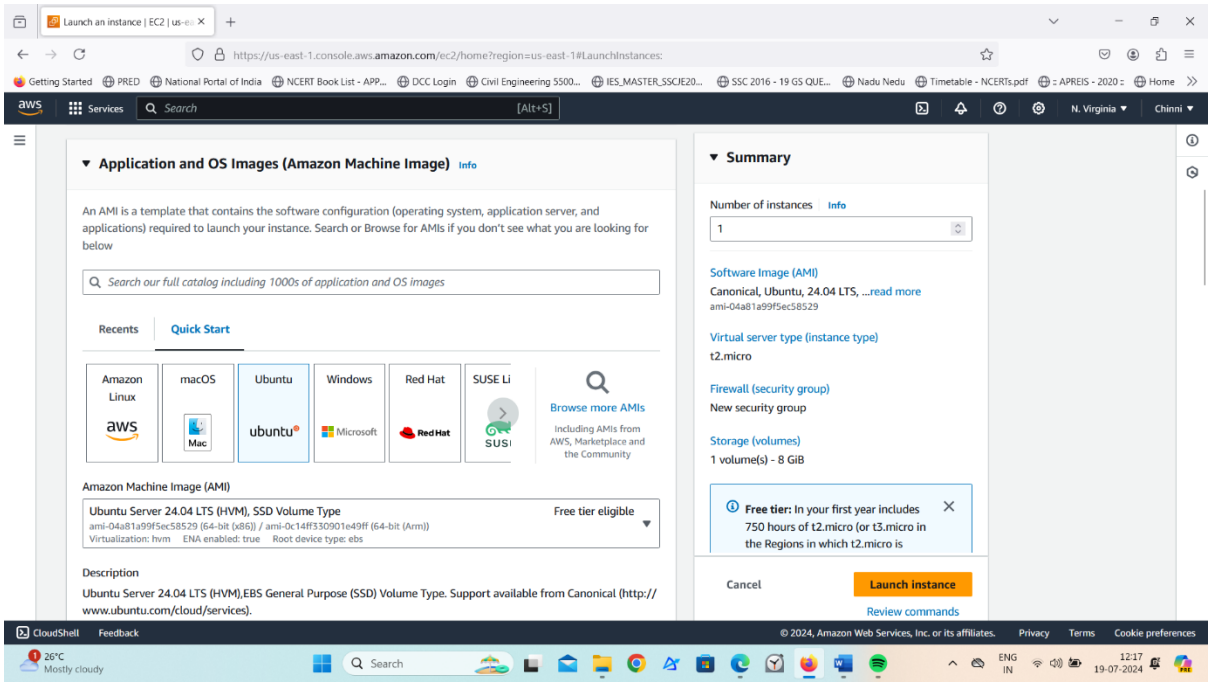
14. **Jenkins** is an open-source automation server used for continuous integration and continuous delivery (CI/CD), facilitating software development processes through automation of build, test, and deployment tasks.

Step1: Launch an ubuntu instance (t2.2xlarge)

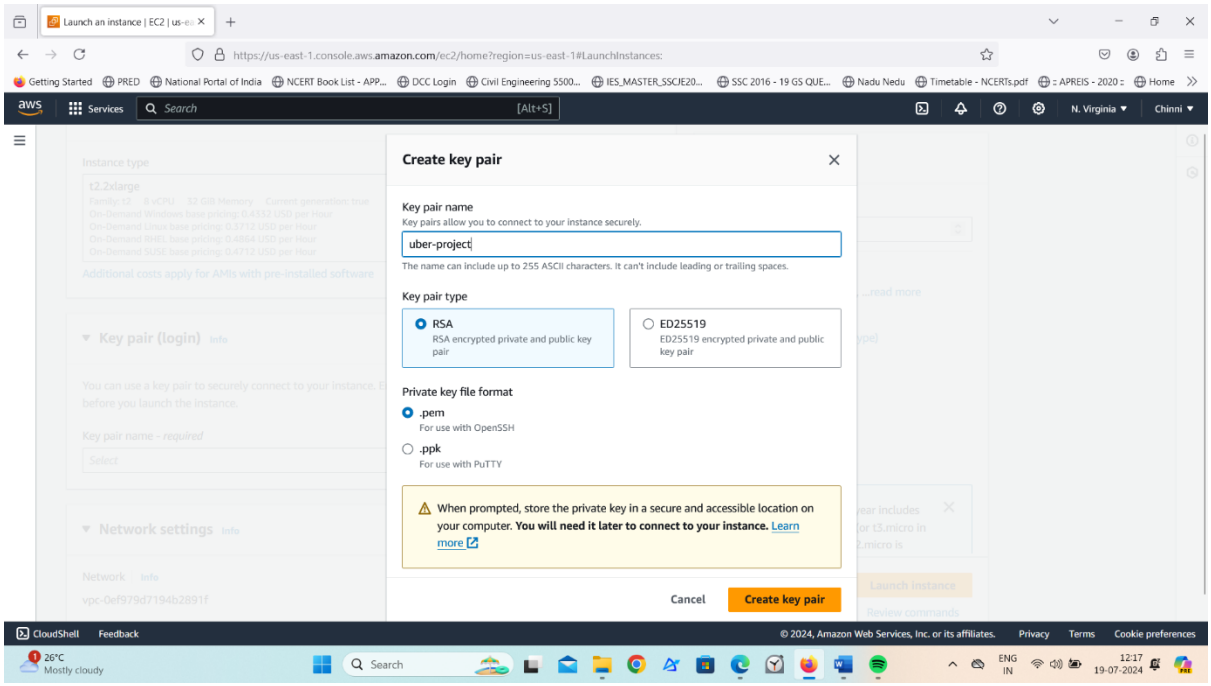
- Sign in to AWS Console: Log in to your AWS Management Console.
- Navigate to EC2 Dashboard.
- Launch Instance: Click on the “Launch Instance” button.



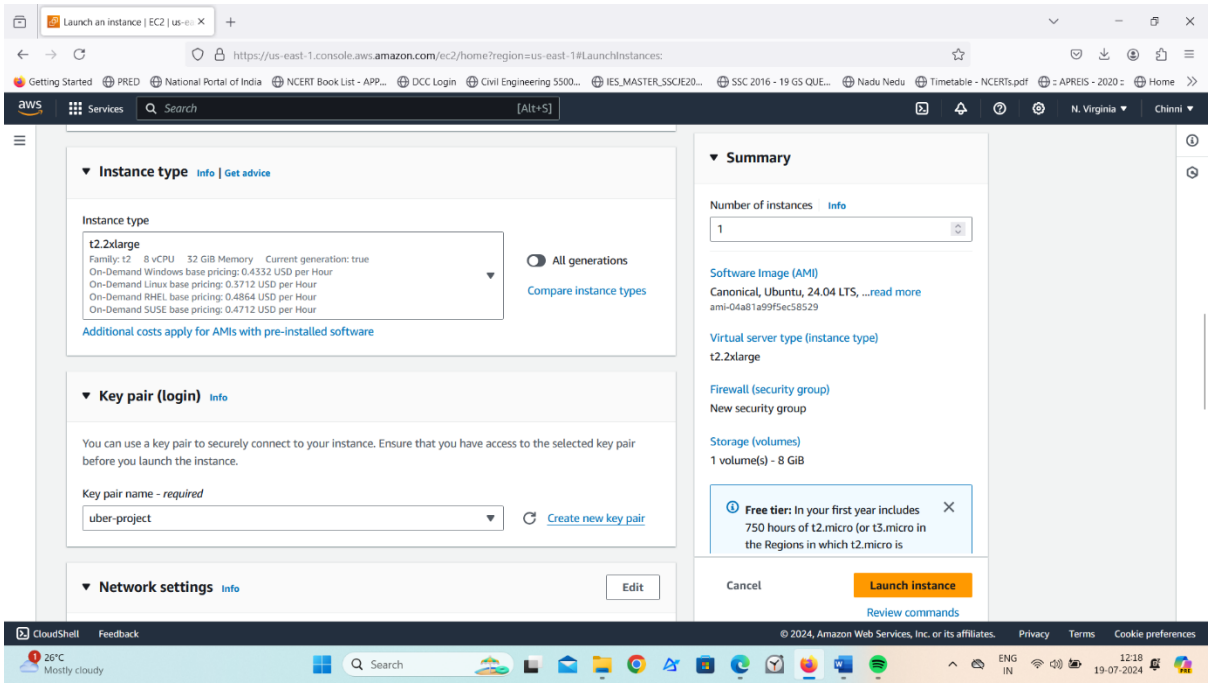
- Choose an Amazon Machine Image (AMI): Select an appropriate AMI for your instance. For example, you can choose Ubuntu image.



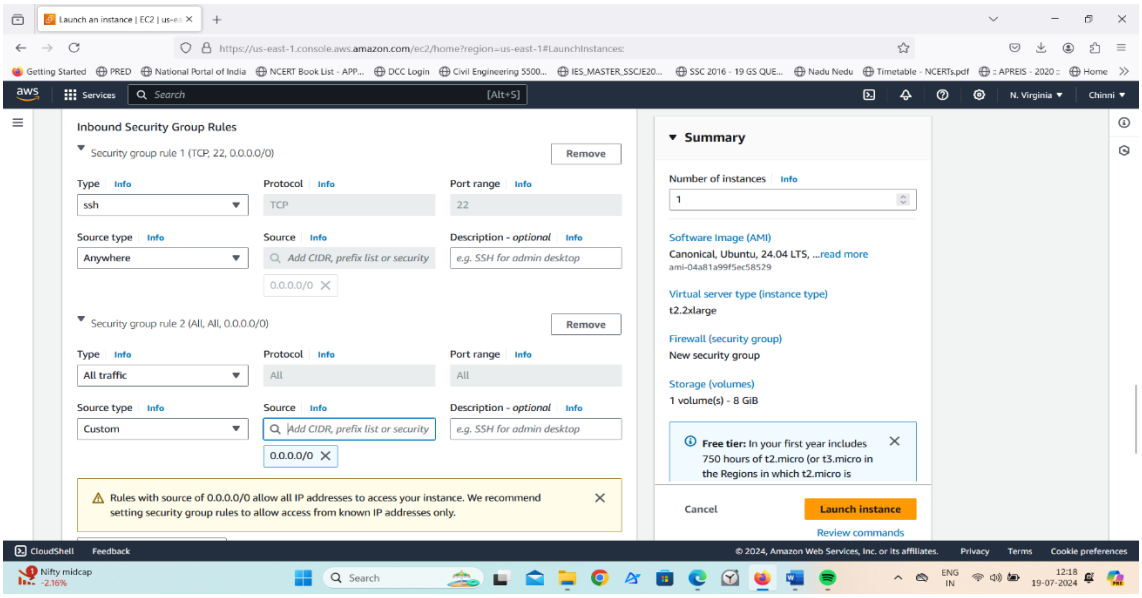
- Create new key pair.

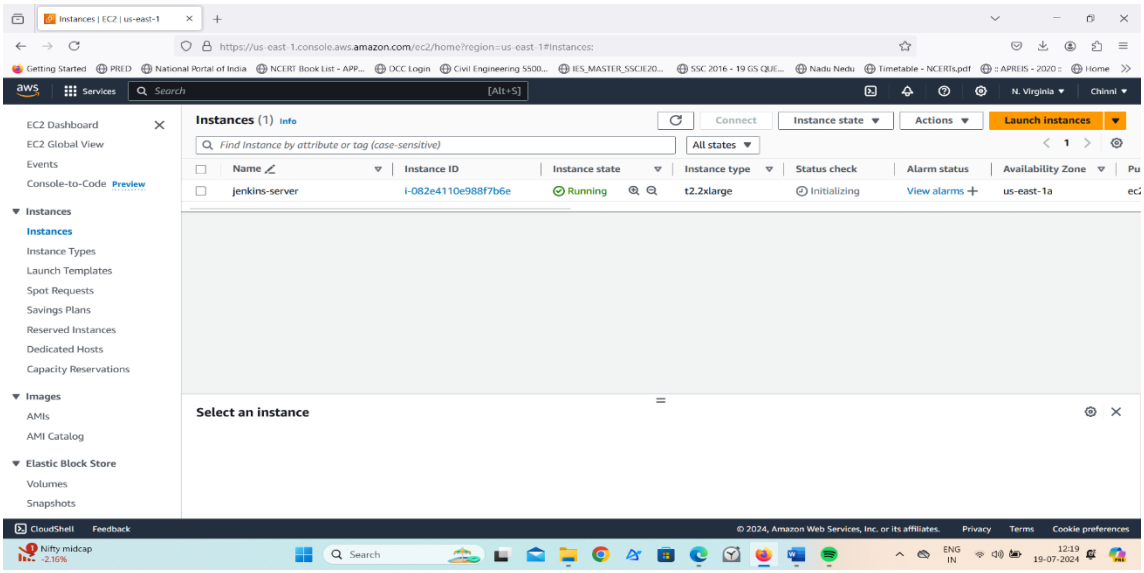


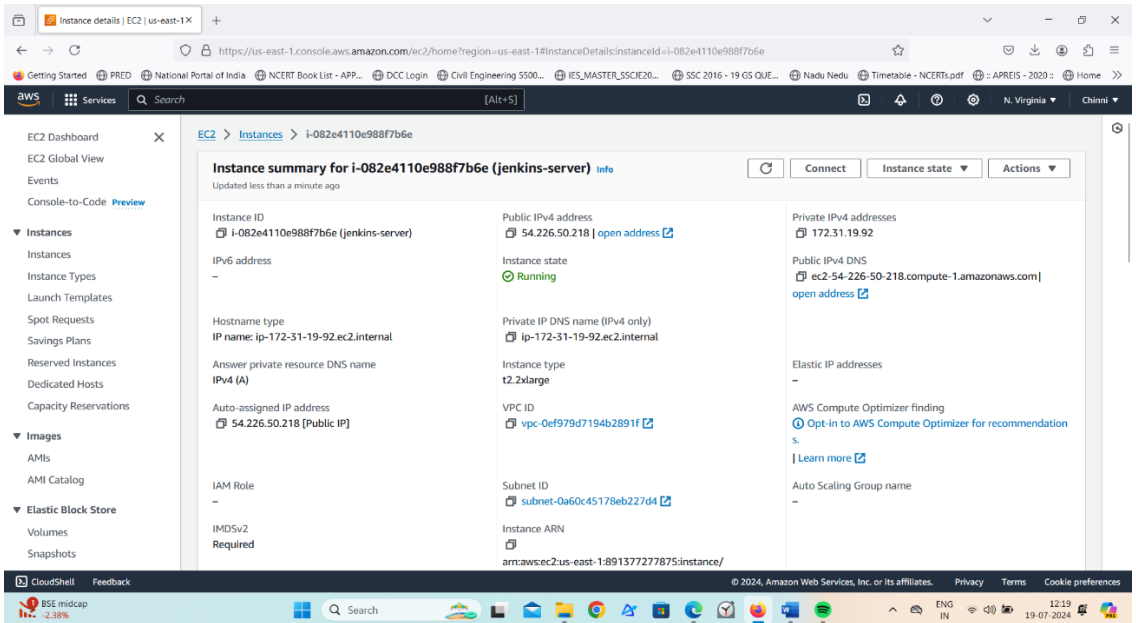
- Choose an Instance Type: In the “Choose Instance Type” step, select t2.2xlarge as your instance type. Proceed by clicking “Next: Configure Instance Details.”



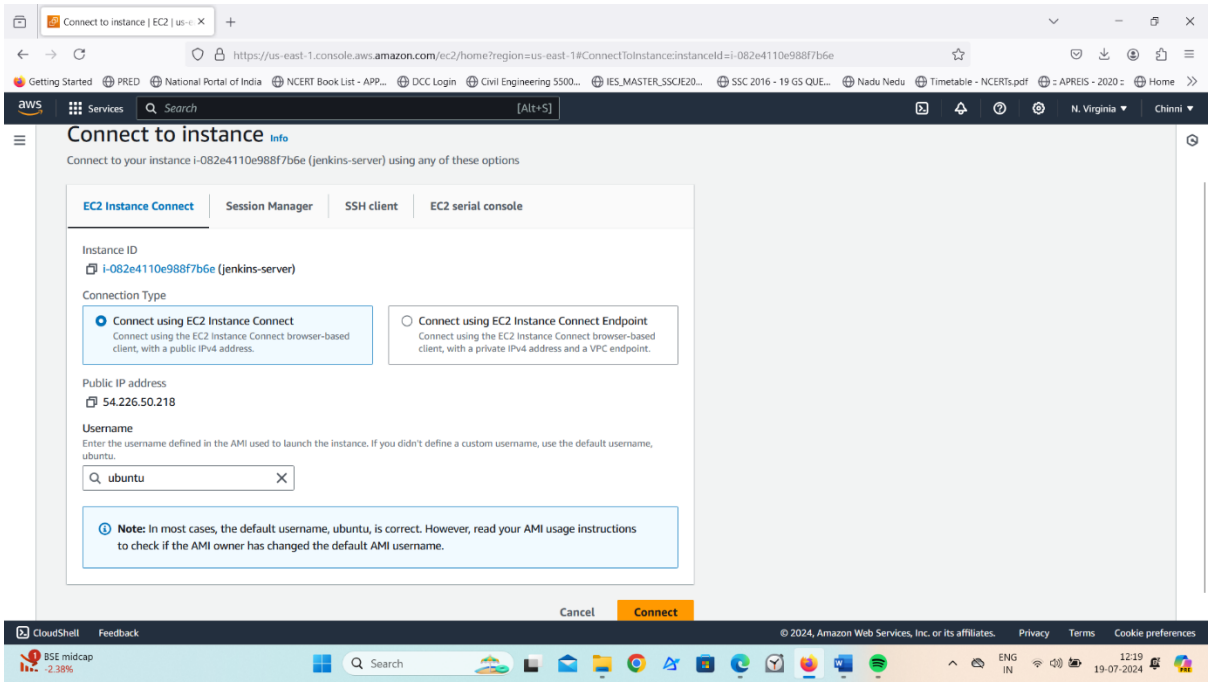
- Configure additional settings like network, subnets, security groups and allow inbound rule as all traffic (0.0.0.0/0)


Launch an instance | EC2 | us-east-1
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#LaunchInstances:
Inbound Security Group Rules
Security group rule 1 (TCP, 22, 0.0.0.0/0)
Remove
Type Info
ssh
Protocol Info
TCP
Port range Info
22
Source type Info
Anywhere
Source Info
Add CIDR, prefix list or security
Description - optional Info
e.g. SSH for admin desktop
0.0.0.0/0
Security group rule 2 (All, All, 0.0.0.0/0)
Remove
Type Info
All traffic
Protocol Info
All
Port range Info
All
Source type Info
Custom
Source Info
Add CIDR, prefix list or security
Description - optional Info
e.g. SSH for admin desktop
0.0.0.0/0
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.
Summary
Number of instances Info
1
Software Image (AMI)
Canonical, Ubuntu, 24.04 LTS, ...read more
ami-04a81a99f5ec58529
Virtual server type (instance type)
t2.2xlarge
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is
Cancel
Launch instance
Review commands
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Instances | EC2 | us-east-1
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Instances (1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
jenkins-server
i-082e4110e988f7b6e
Running
t2.2xlarge
Initializing
View alarms
us-east-1a
Select an instance
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

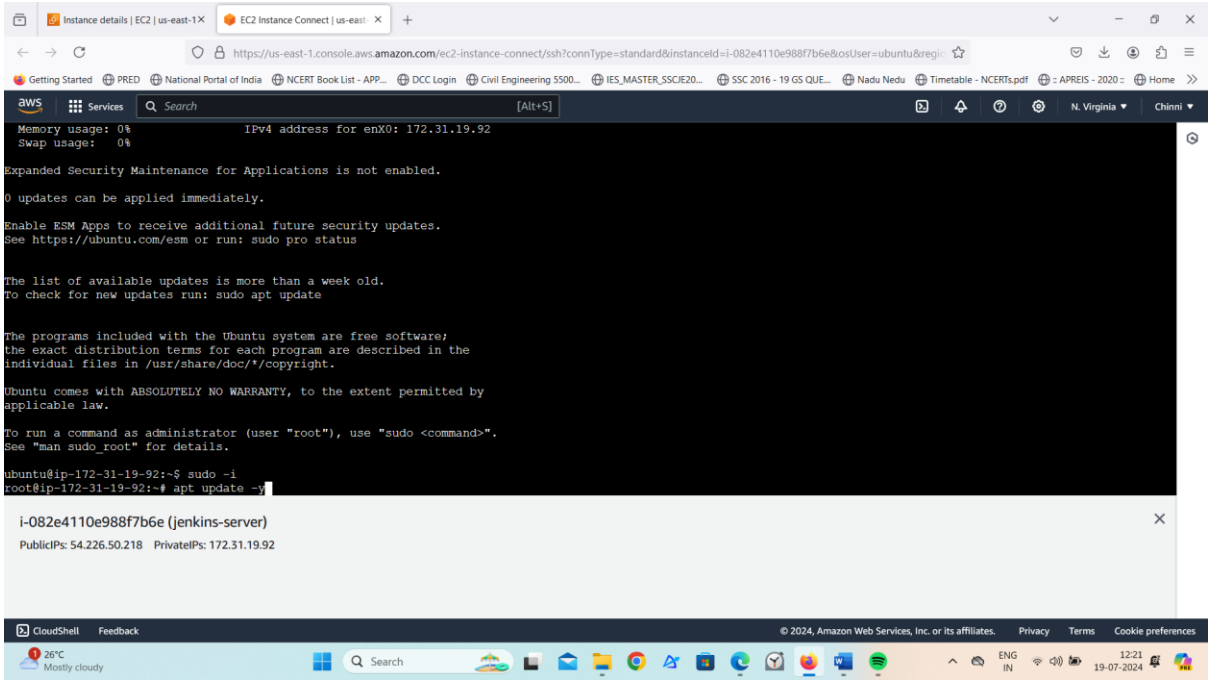

Instance details | EC2 | us-east-1
https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-082e4110e988f7b6e
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
EC2 > Instances > i-082e4110e988f7b6e
Instance summary for i-082e4110e988f7b6e (jenkins-server) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-082e4110e988f7b6e (jenkins-server)
Public IPv4 address
54.226.50.218 | open address
Private IPv4 addresses
172.31.19.92
IPv6 address
–
Instance state
Running
Public IPv4 DNS
ec2-54-226-50-218.compute-1.amazonaws.com | open address
Hostname type
IP name: ip-172-31-19-92.ec2.internal
Private IP DNS name (IPv4 only)
ip-172-31-19-92.ec2.internal
Answer private resource DNS name
IPv4 (A)
Instance type
t2.2xlarge
Elastic IP addresses
–
Auto-assigned IP address
54.226.50.218 [Public IP]
VPC ID
vpc-0ef979d7194b2891f
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations.
Learn more
IAM Role
–
Subnet ID
subnet-0a60c45178eb227d4
Auto Scaling Group name
–
IMDSv2
Required
Instance ARN
arn:aws:ec2:us-east-1:891377277875:instance/
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Connect to the instance.



sudo -i → convert to root user

update the server → apt update -y

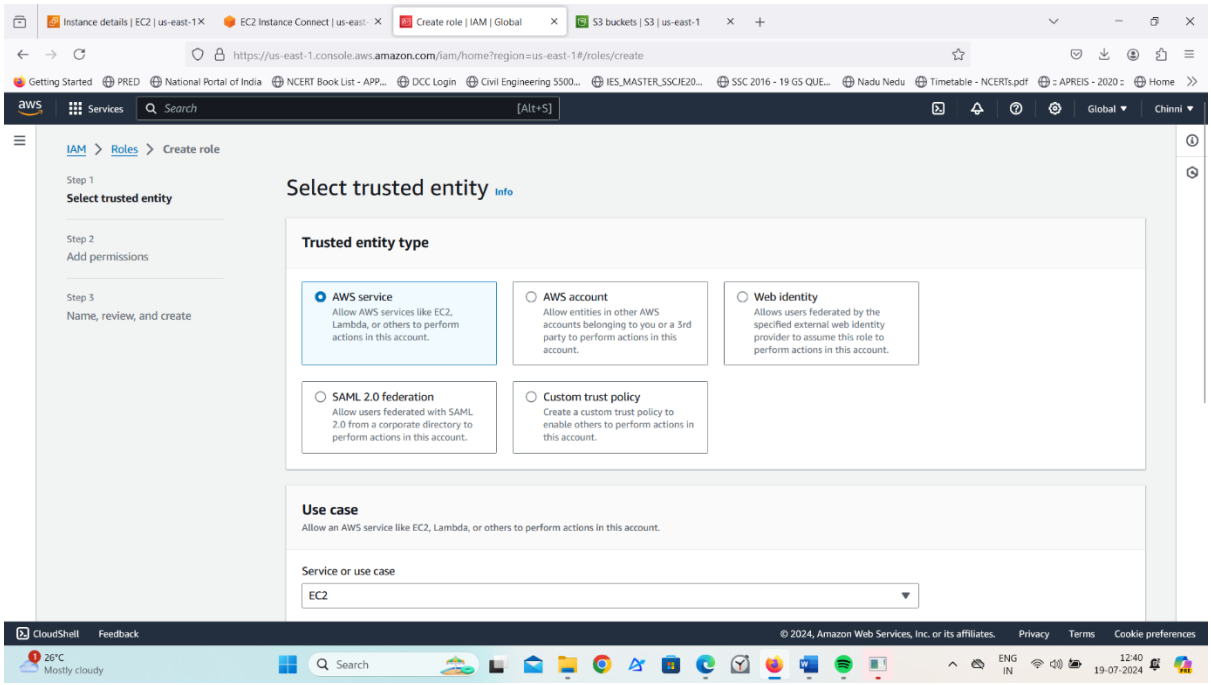


Step2: Create IAM role

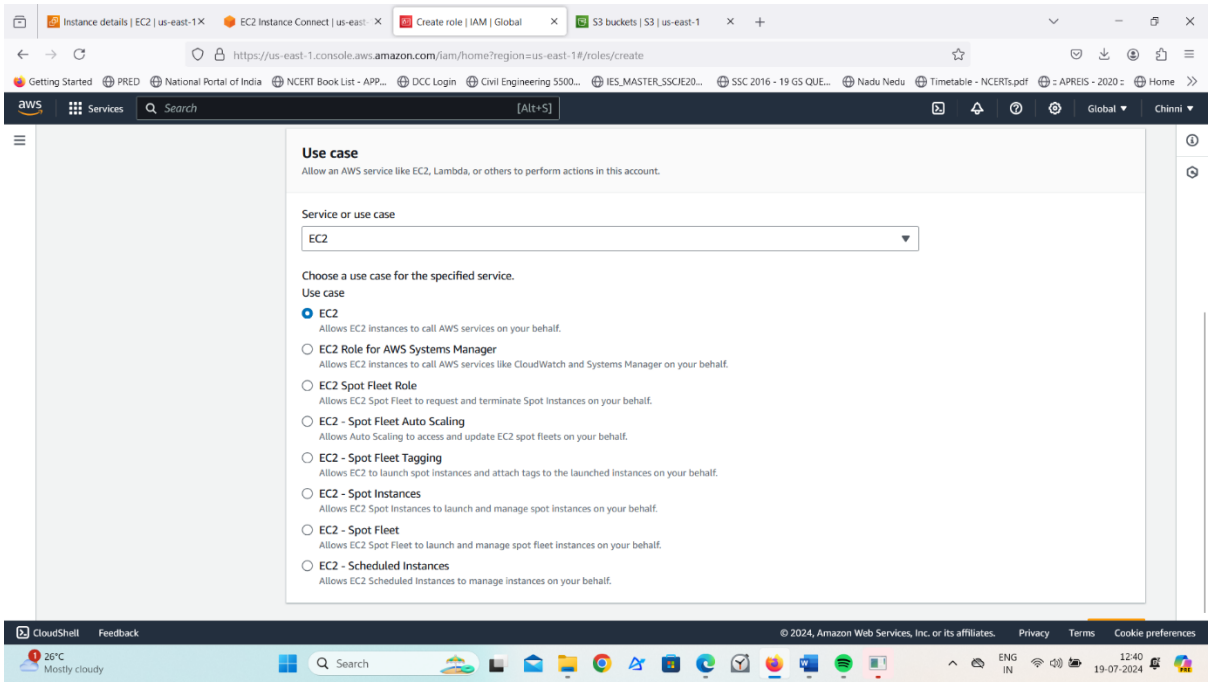
Search for IAM in the search bar of AWS and click on roles.

Click on Create Role

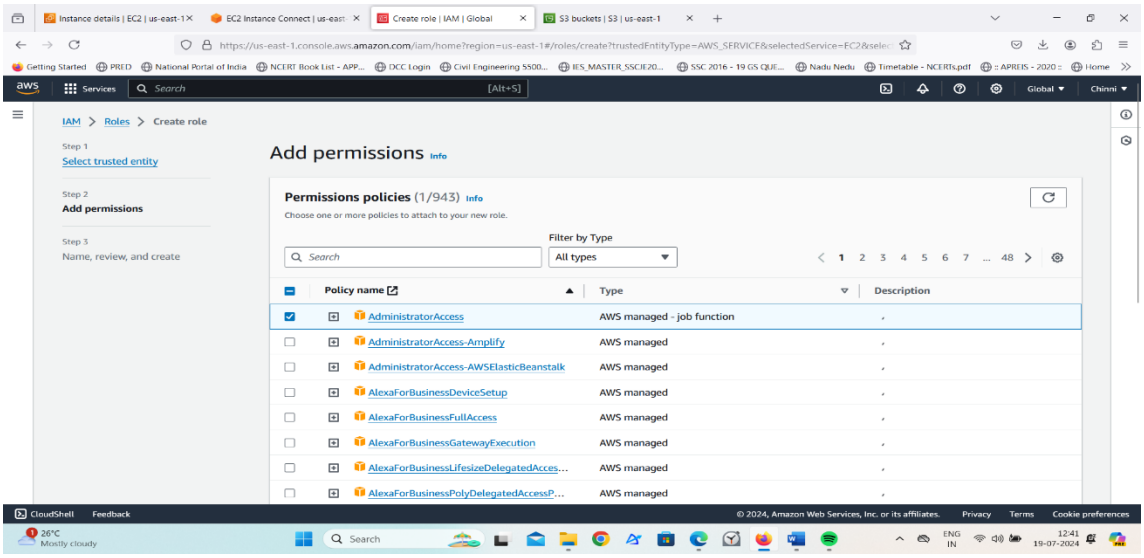
Select entity type as AWS service



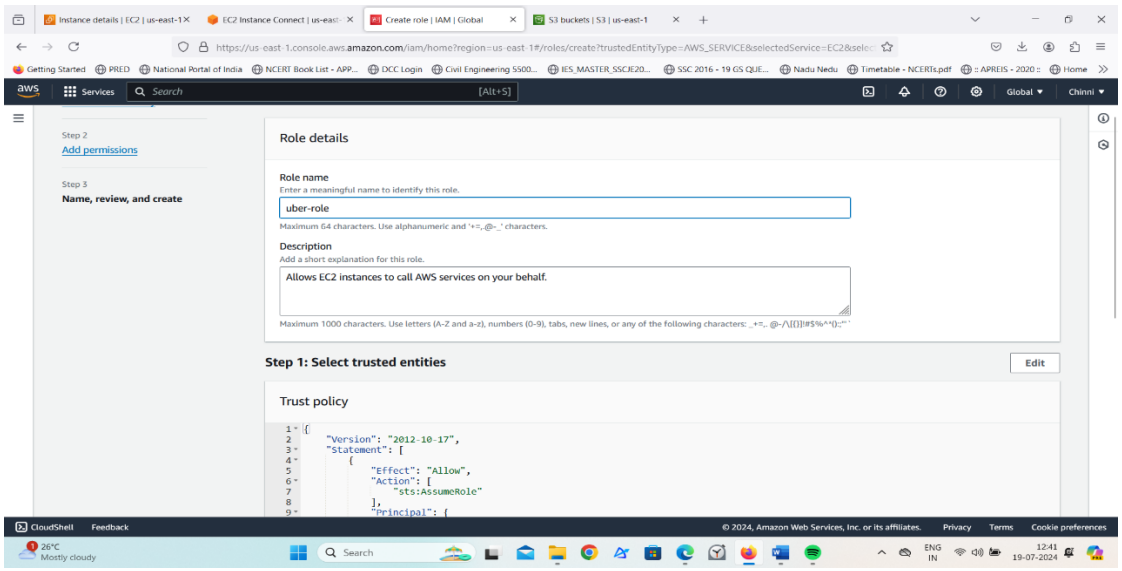
Use case as EC2 and click on Next.

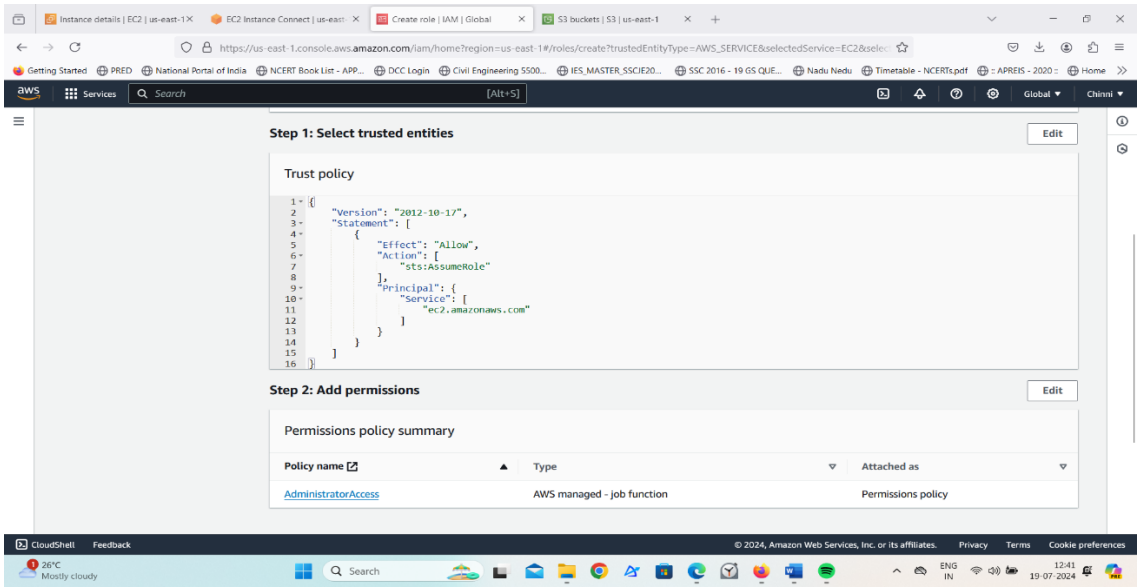


For permission policy select Administrator Access, click Next.

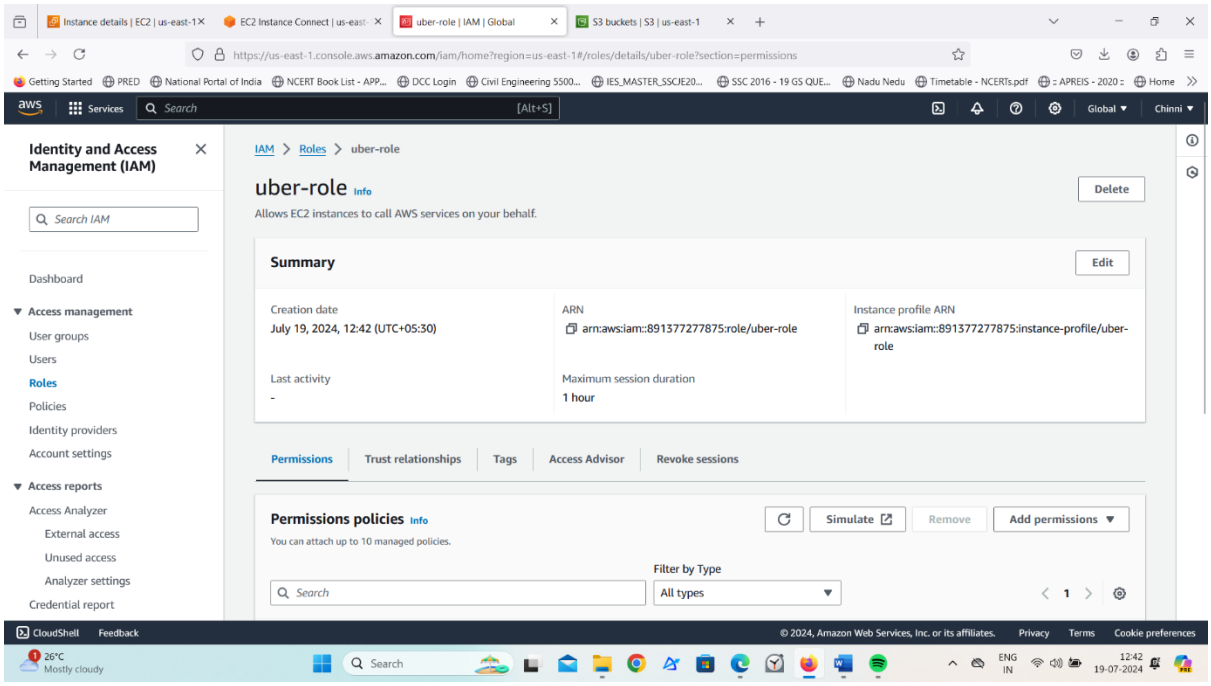


Provide a Name for Role and click on Create role.





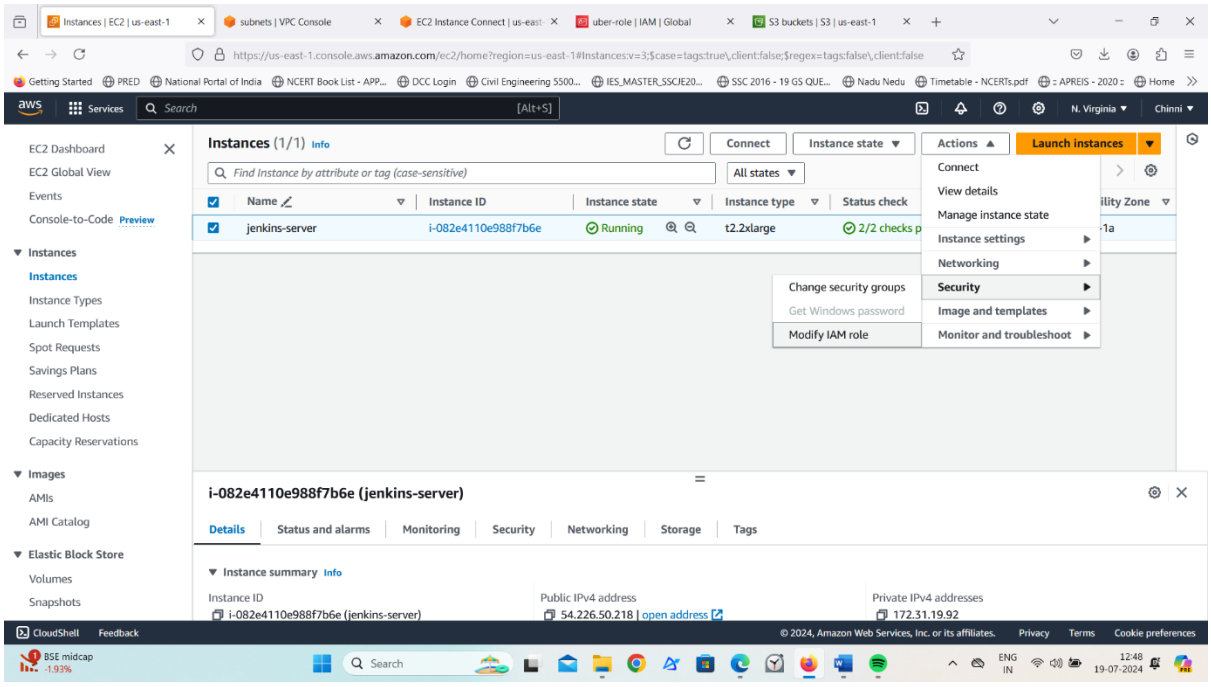
Role is created.



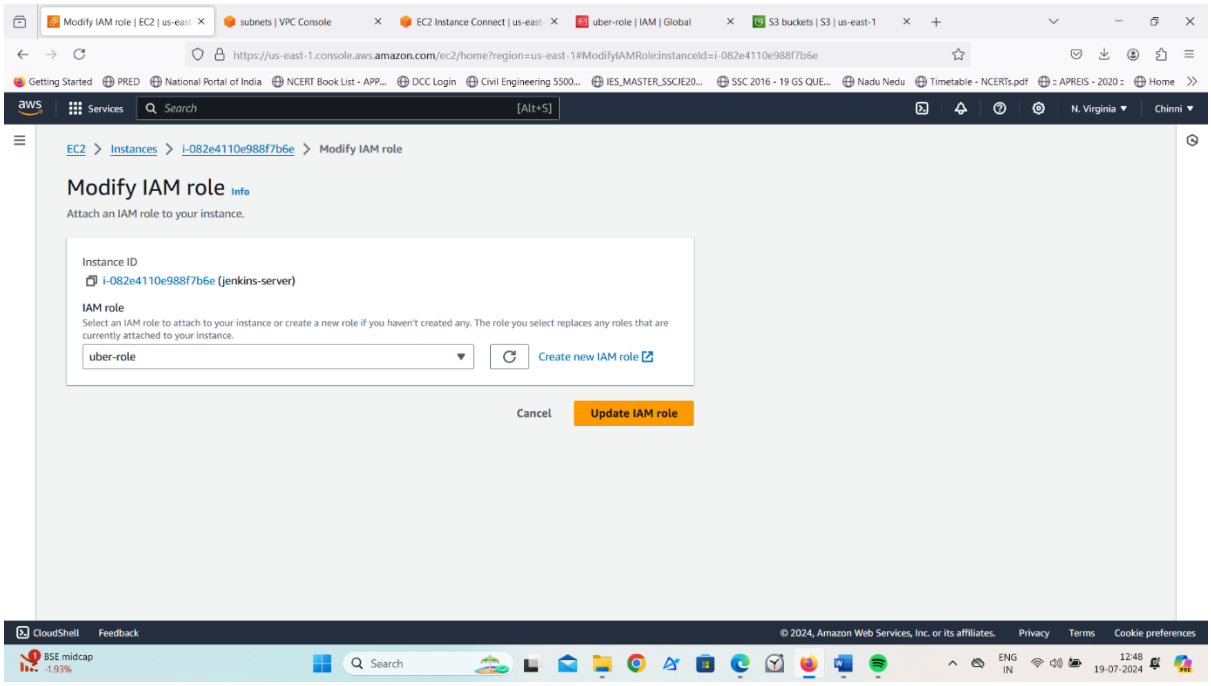
Now Attach this role to Ec2 instance that we created earlier, so we can provision cluster from that instance.

Go to EC2 Dashboard and select the instance.

Click on Actions → Security → Modify IAM role.



Select the Role that created earlier and click on Update IAM role.



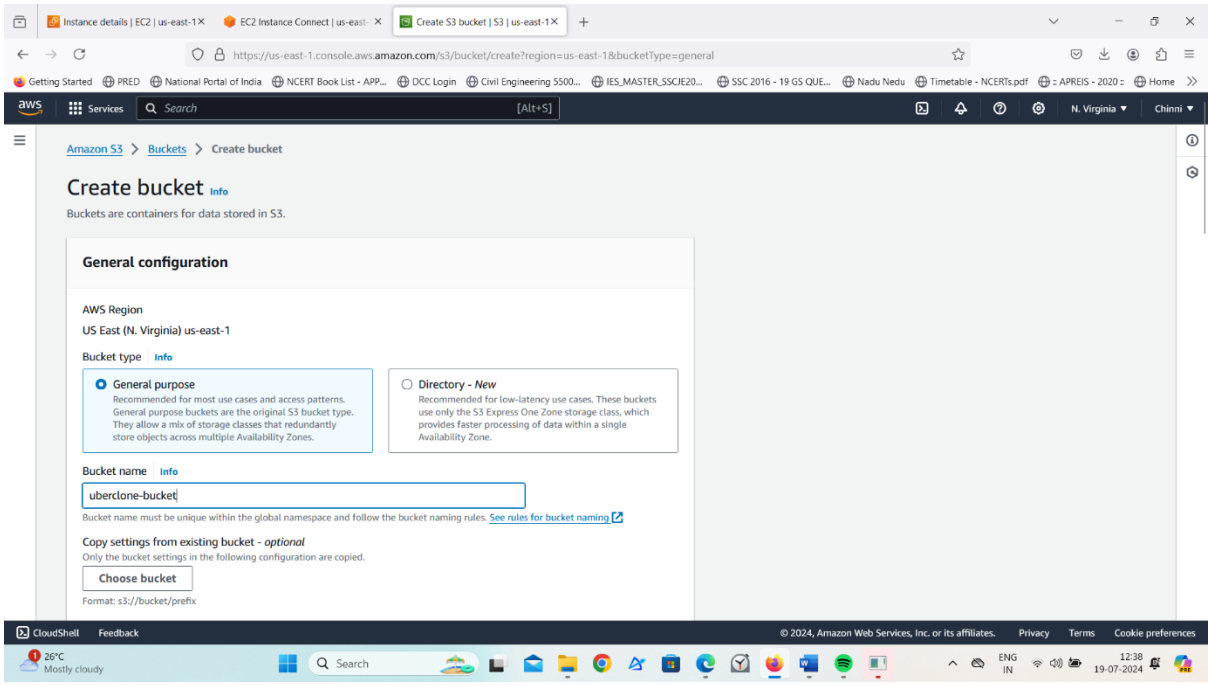
Connect to the instance.

Step3: Create S3 bucket

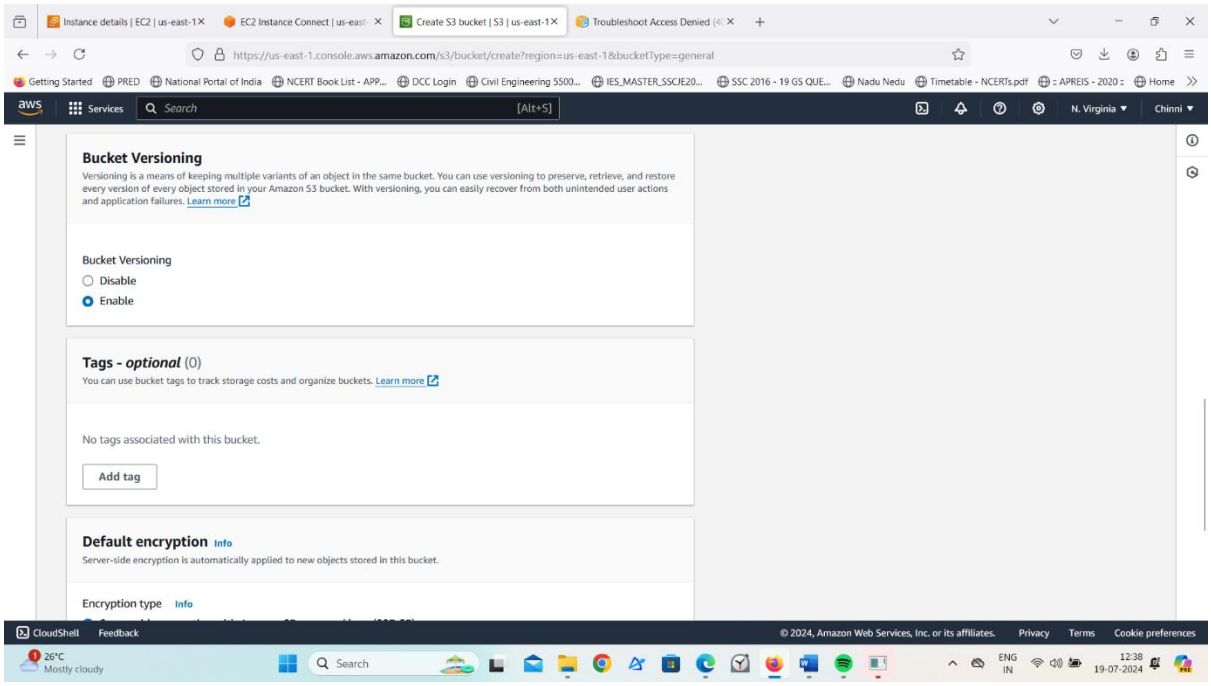
Search for S3 bucket in the search bar of AWS and click on it.

Click on S3 bucket.

Choose bucket type as general purpose and give bucket name.

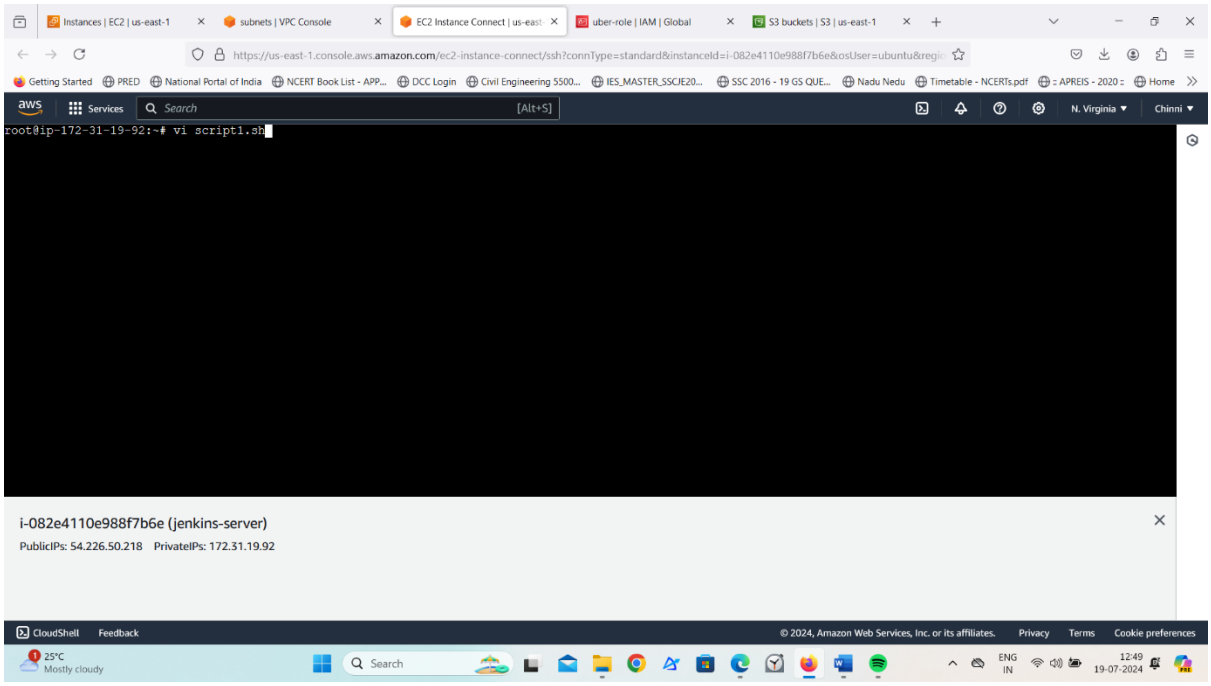


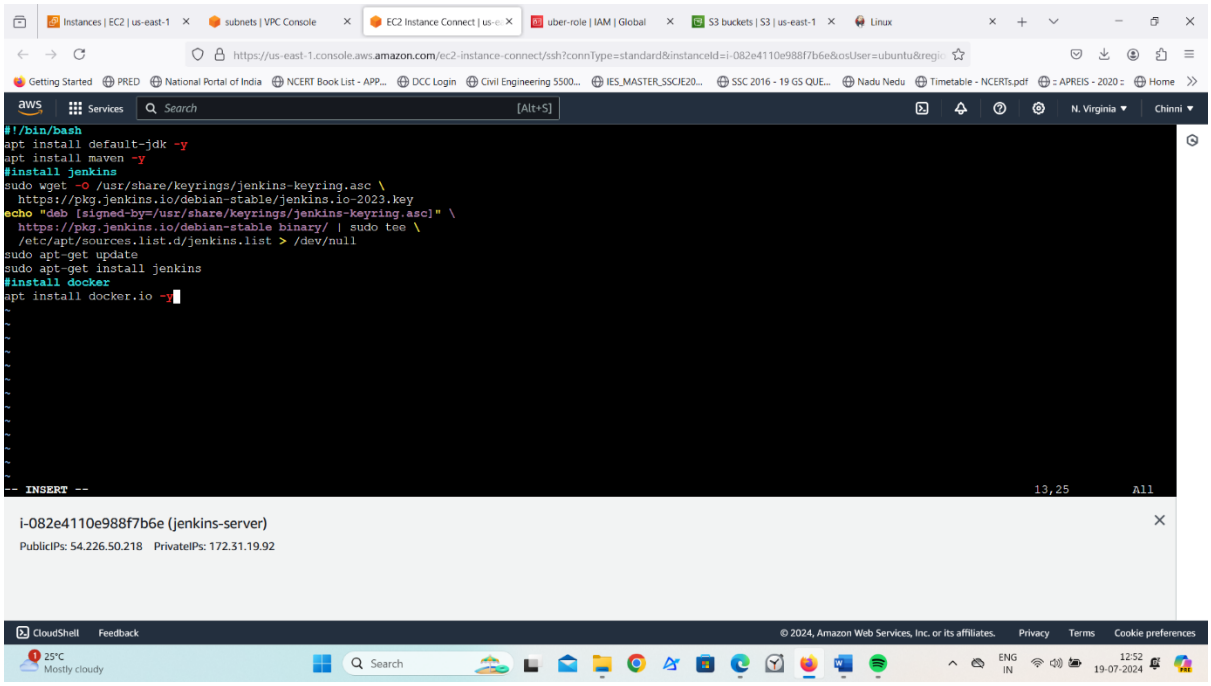
Enable bucket versioning. Click on create bucket.



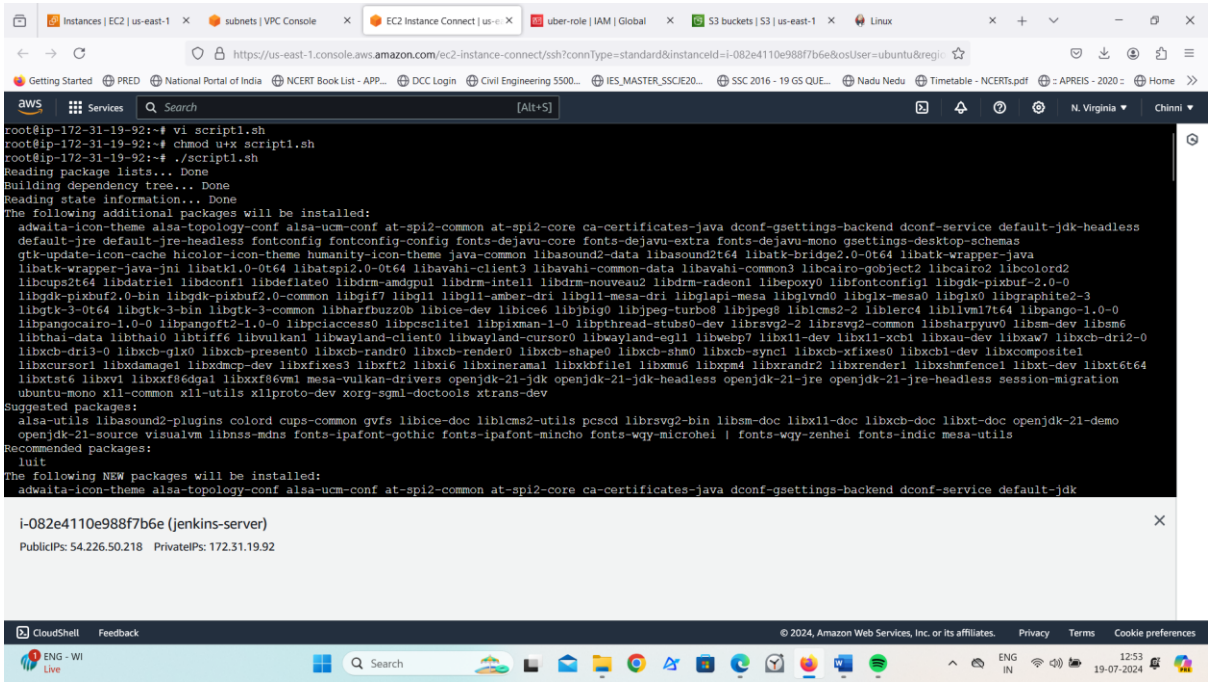
Step4: Installation of packages

- Create shell script in our server. Using vi script1.sh
- Enter this script into it. This script installs Java, Jenkins, maven, Docker.





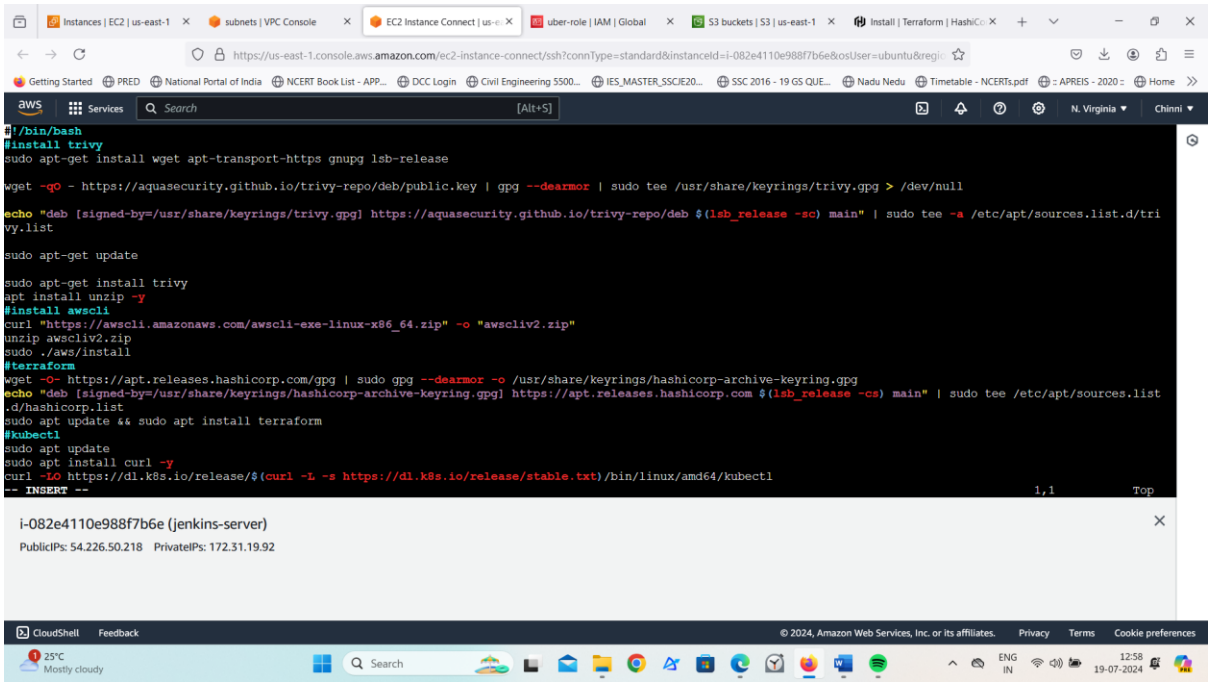
For that file change the file permission, using chmod. And execute the file.



Create another script file using vi script2.sh

This script installs Kubectl, Terraform, AWS Cli, SonarQube, Trivy.

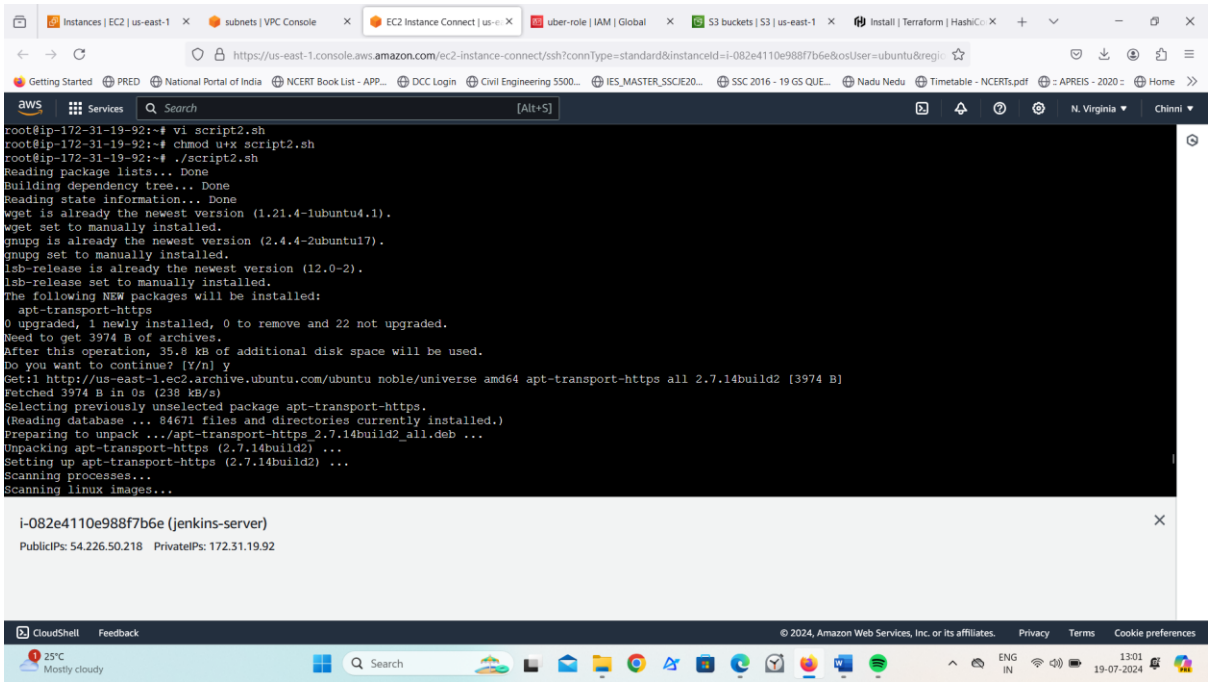
For that file change the file permission, using chmod. And execute the file.



```
#!/bin/bash
#install trivy
sudo apt-get install wget apt-transport-https gnupg lsb-release

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/tri
vy.list

sudo apt-get update

sudo apt-get install trivy
apt install unzip -y
#install awscli
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
#terraform
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list
.d/hashicorp.list
sudo apt update && sudo apt install terraform
#kubectl
sudo apt update
sudo apt install curl -y
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
-- INSERT --                                                                    1,1           Top
```

i-082e4110e988f7b6e (jenkins-server)

PublicIPs: 54.226.50.218 PrivateIPs: 172.31.19.92



```
root@ip-172-31-19-92:~# vi script2.sh
root@ip-172-31-19-92:~# chmod u+x script2.sh
root@ip-172-31-19-92:~# ./script2.sh
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wget is already the newest version (1.21.4-1ubuntu4.1).
wget set to manually installed.
gnupg is already the newest version (2.4.4-2ubuntu17).
gnupg set to manually installed.
lsb-release is already the newest version (12.0-2).
lsb-release set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 22 not upgraded.
Need to get 3974 B of archives.
After this operation, 35.8 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 apt-transport-https all 2.7.14build2 [3974 B]
Fetched 3974 B in 0s (238 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 84671 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
Unpacking apt-transport-https (2.7.14build2) ...
Setting up apt-transport-https (2.7.14build2) ...
Scanning processes...
Scanning linux images...
```

i-082e4110e988f7b6e (jenkins-server)

PublicIPs: 54.226.50.218 PrivateIPs: 172.31.19.92

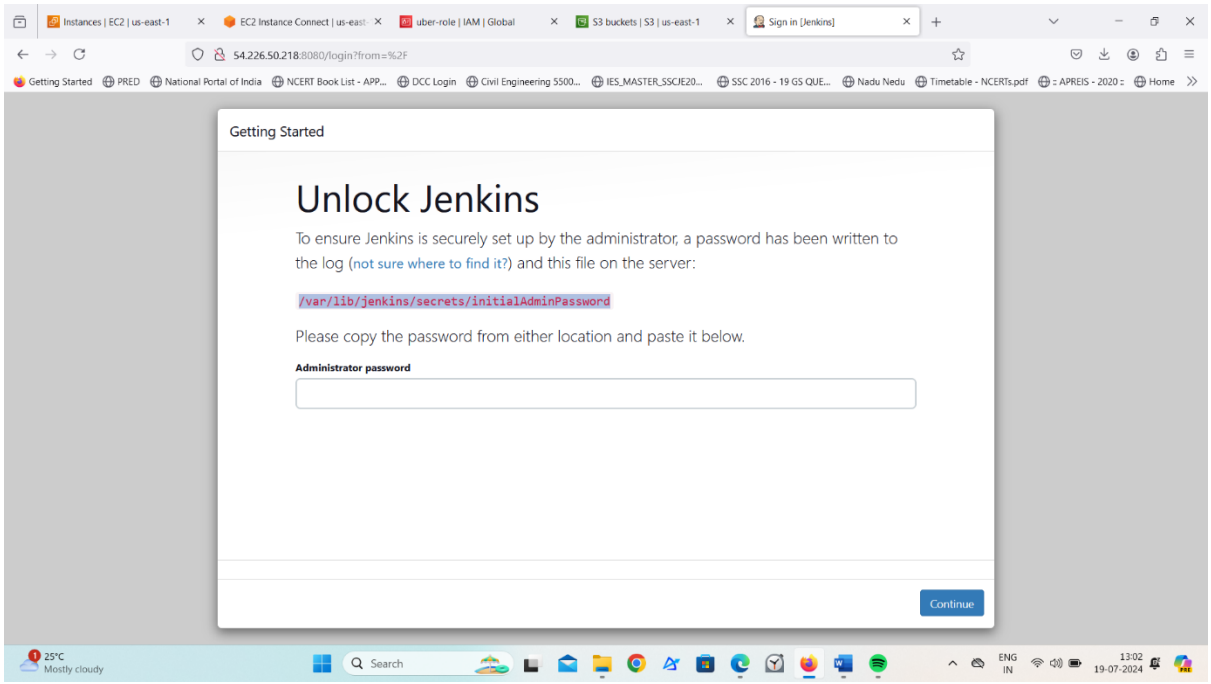
Step5: Connect to Jenkins and SonarQube servers

Now copy the public IP address of ec2 and paste it into the browser

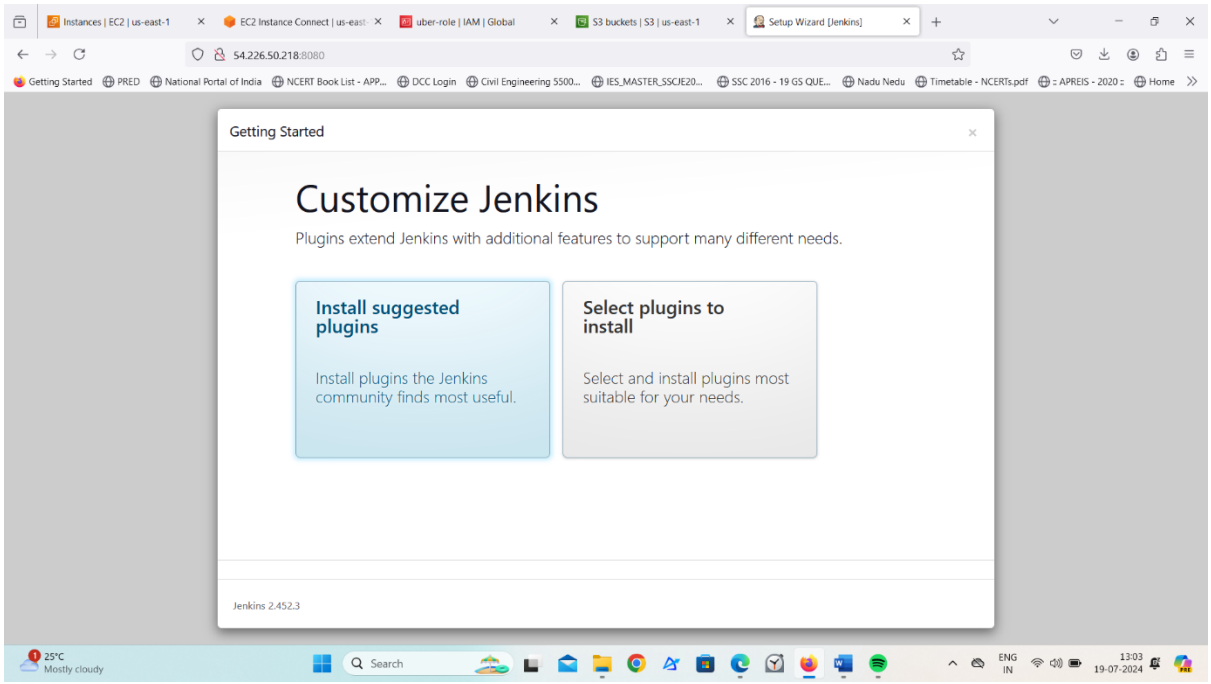
<Ec2-ip:8080> #you will see Jenkins login page

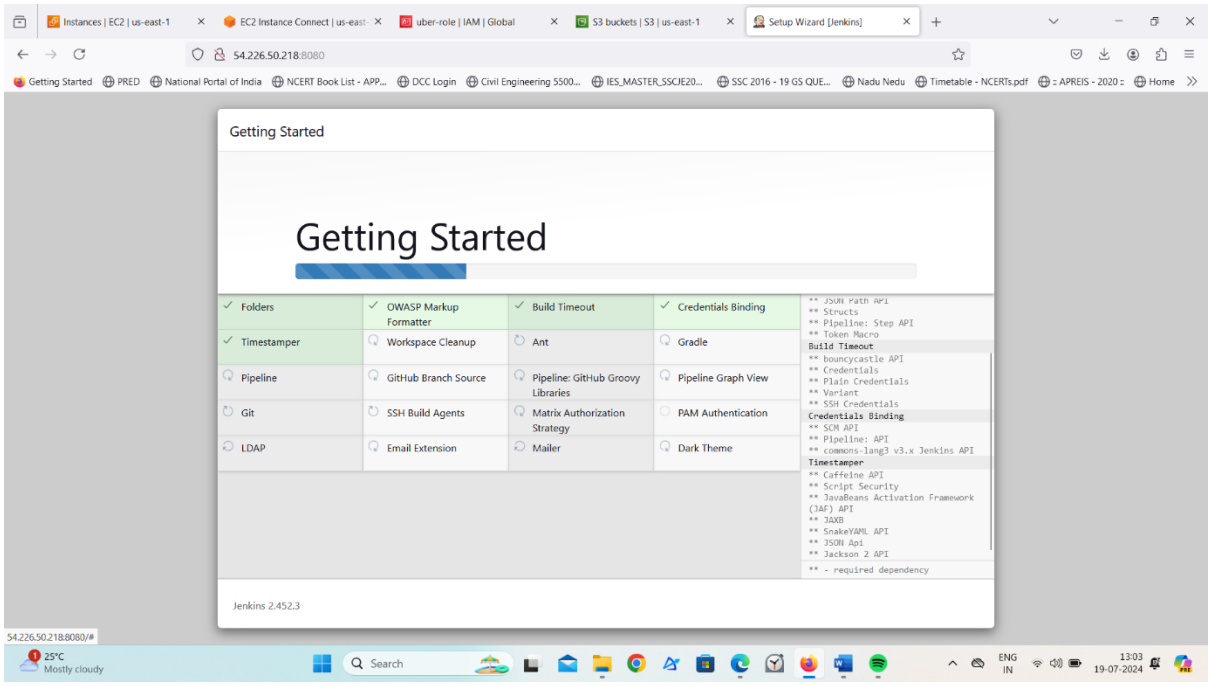
provide the below command for the Administrator password

sudo cat /var/lib/jenkins/secrets/initialAdminPassword



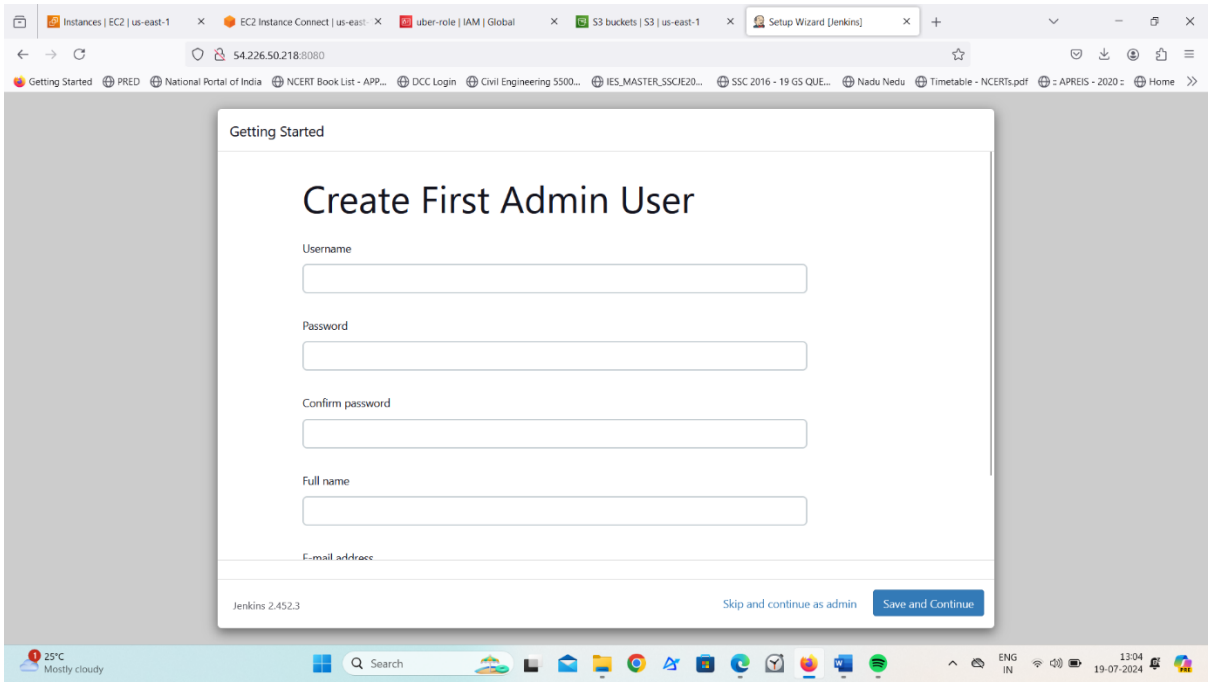
Now, install the suggested plugins.



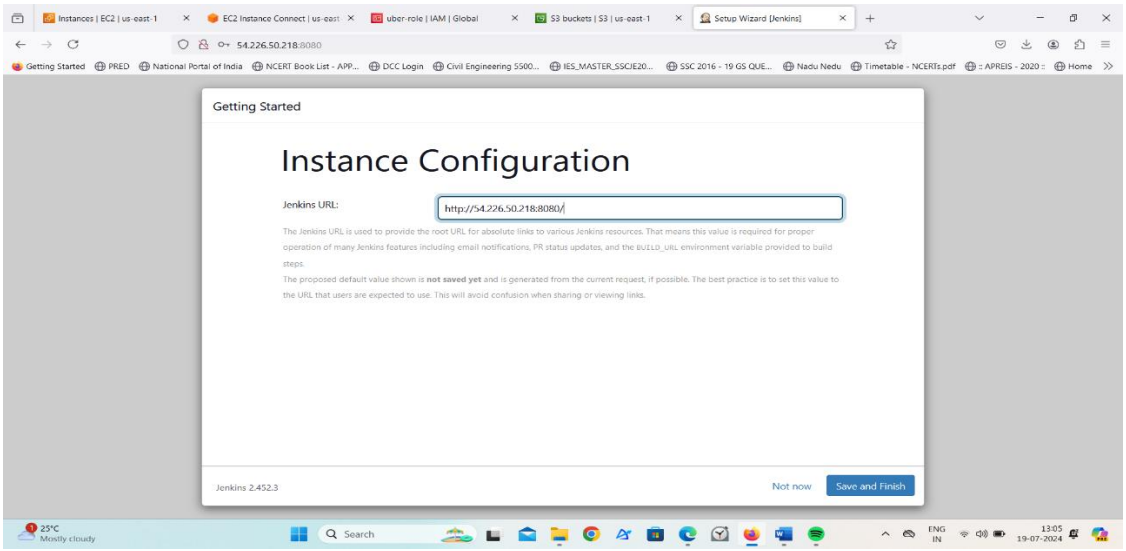


Jenkins will now get installed and install all the libraries.

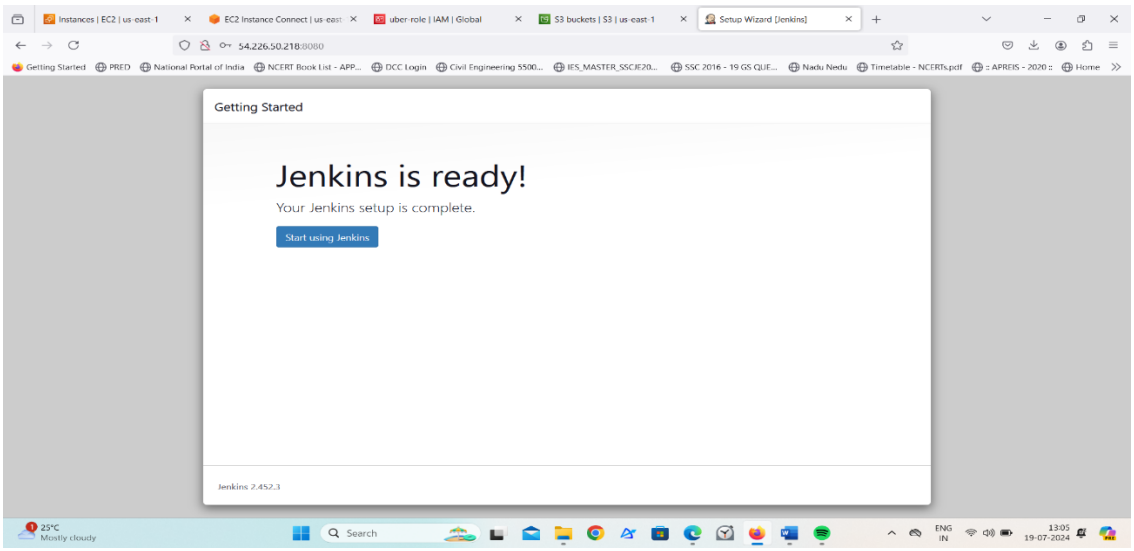
Create an admin user



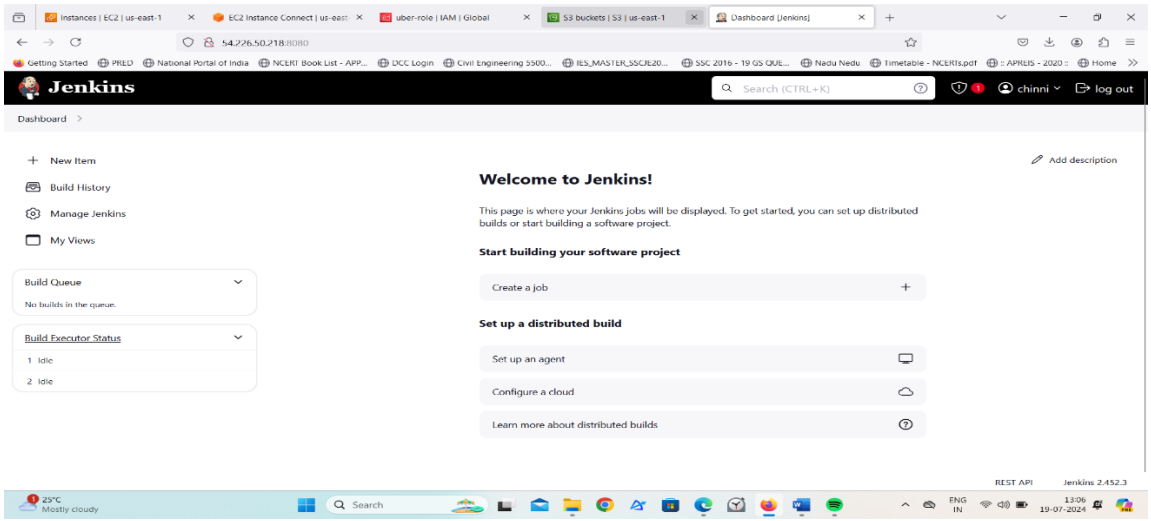
Click on save and continue.



Click on save and Finish



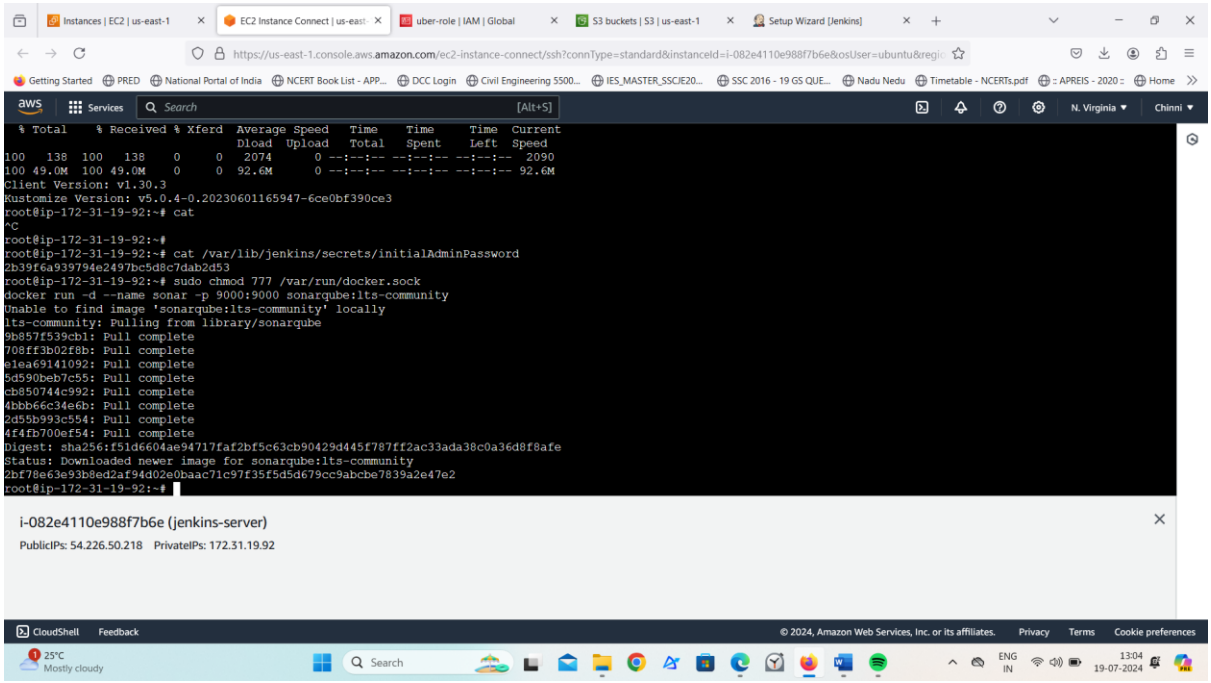
Click on start using Jenkins. We see Jenkins dashboard.



Using docker we will install SonarQube server.

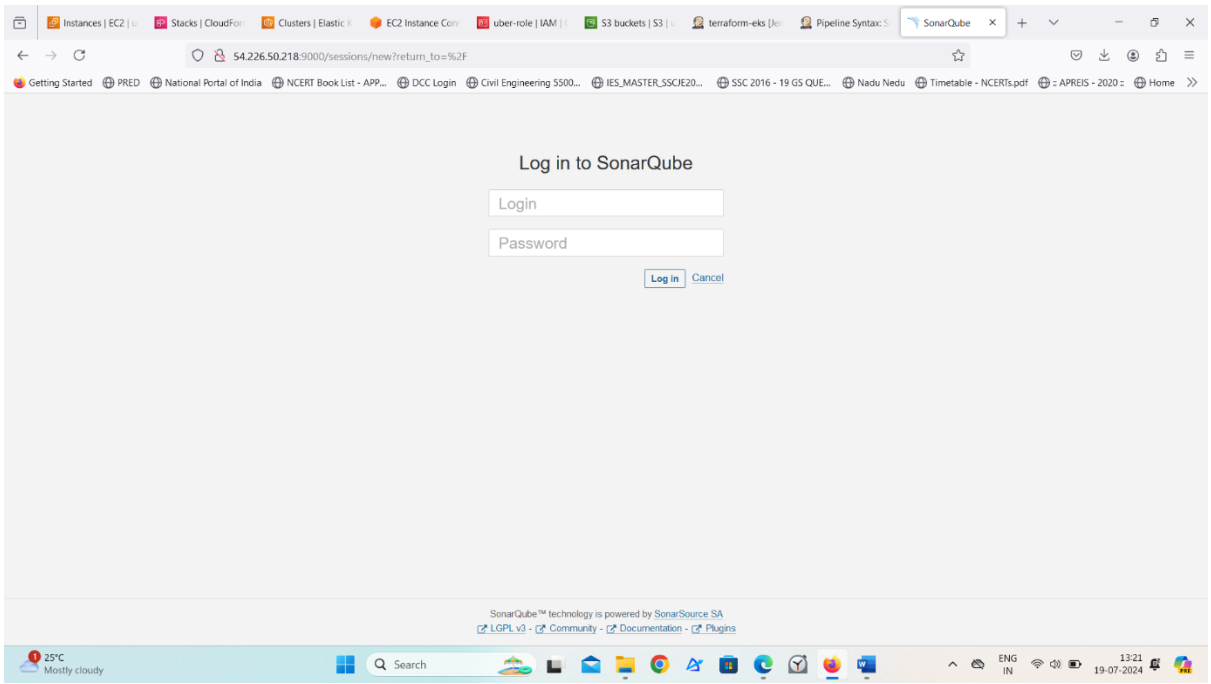
sudo chmod 777 /var/run/docker.sock

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community



Now Copy the public IP again and paste it into a new tab in the browser with 9000

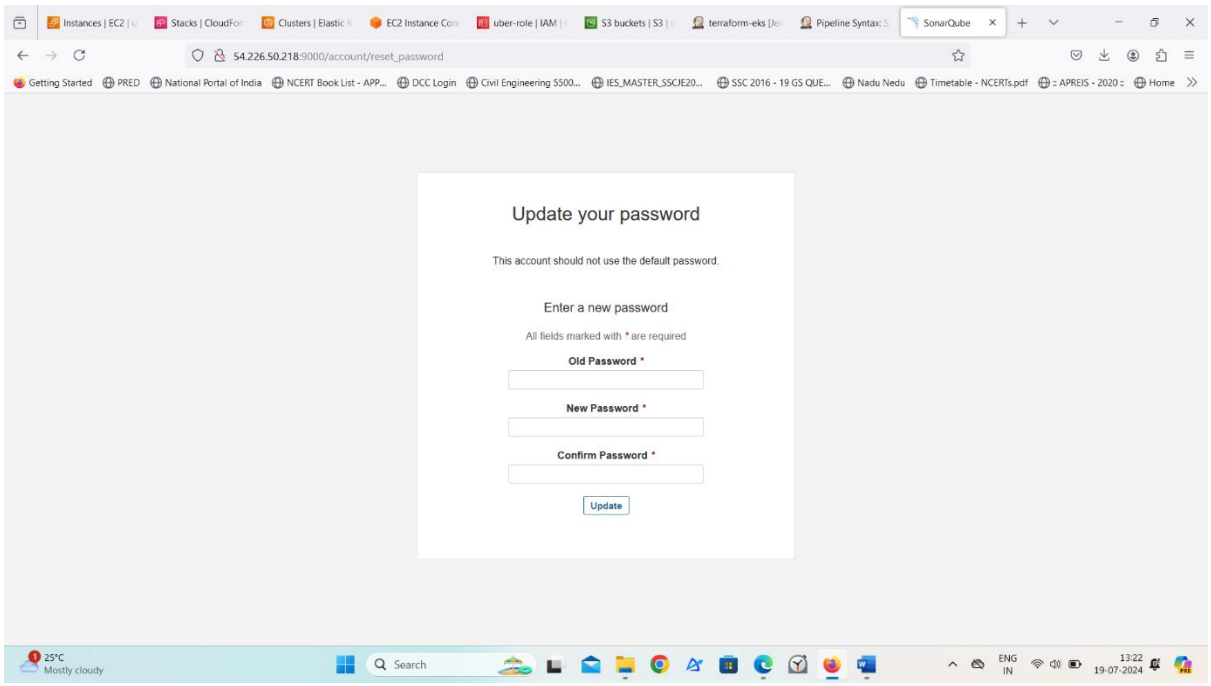
<ec2-ip:9000> #runs sonar container



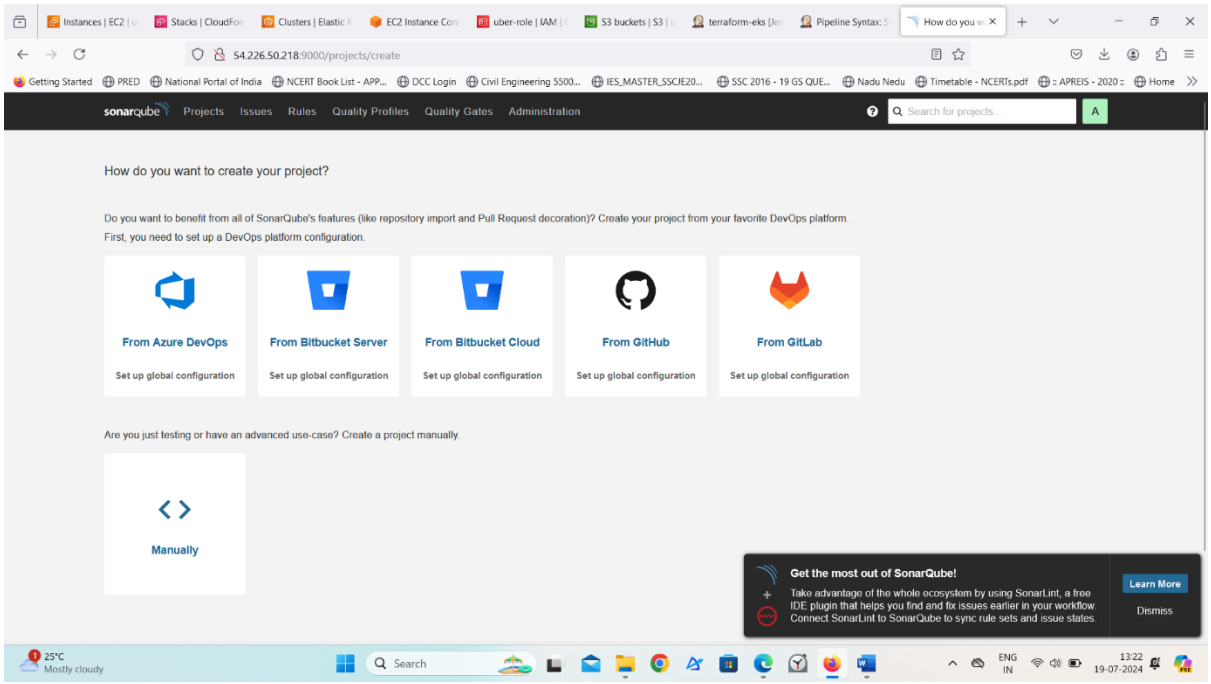
Enter username and password, click on login and change password

username admin

password admin



Update New password, this is Sonar Dashboard.



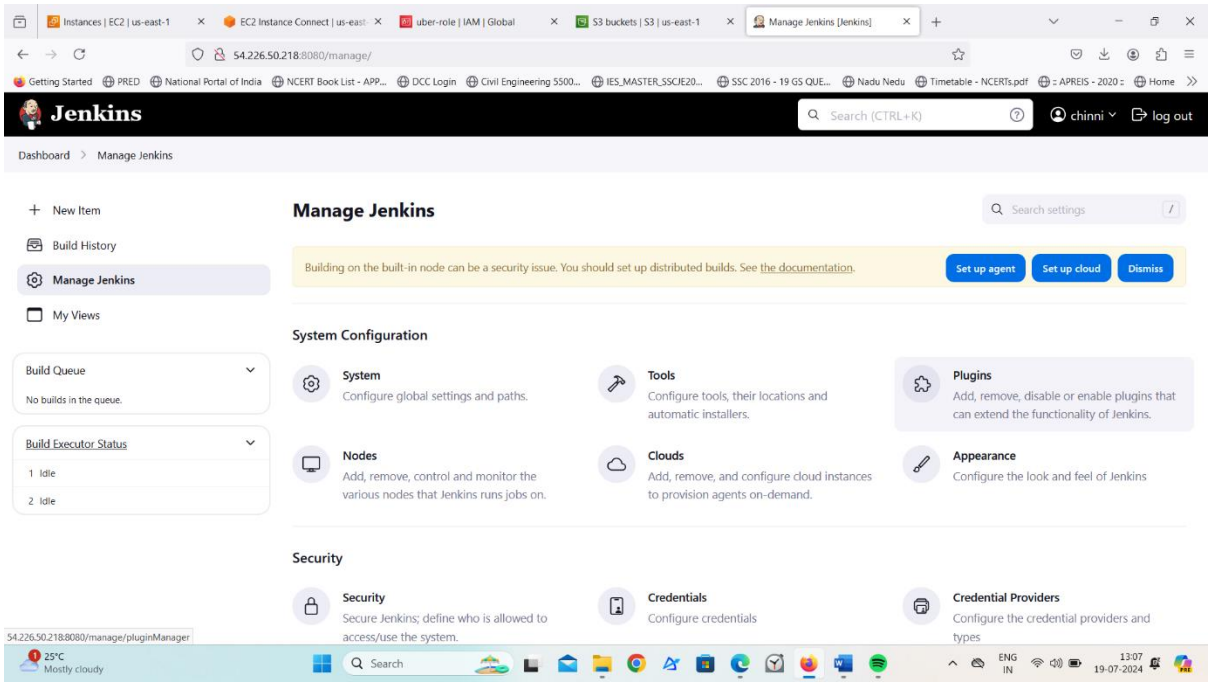
We see sonar server is accessed.

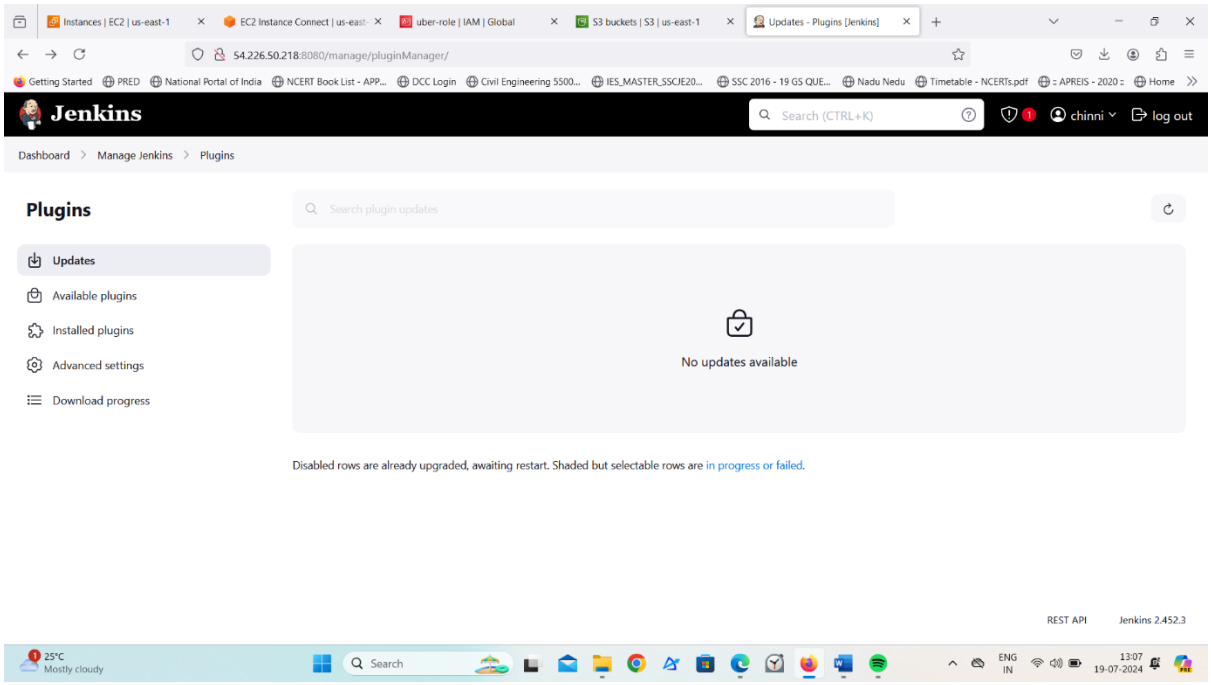
Step6: Terraform plugin install and EKS provision

Now go to Jenkins and add a terraform plugin to provision the AWS EKS using the Pipeline Job.

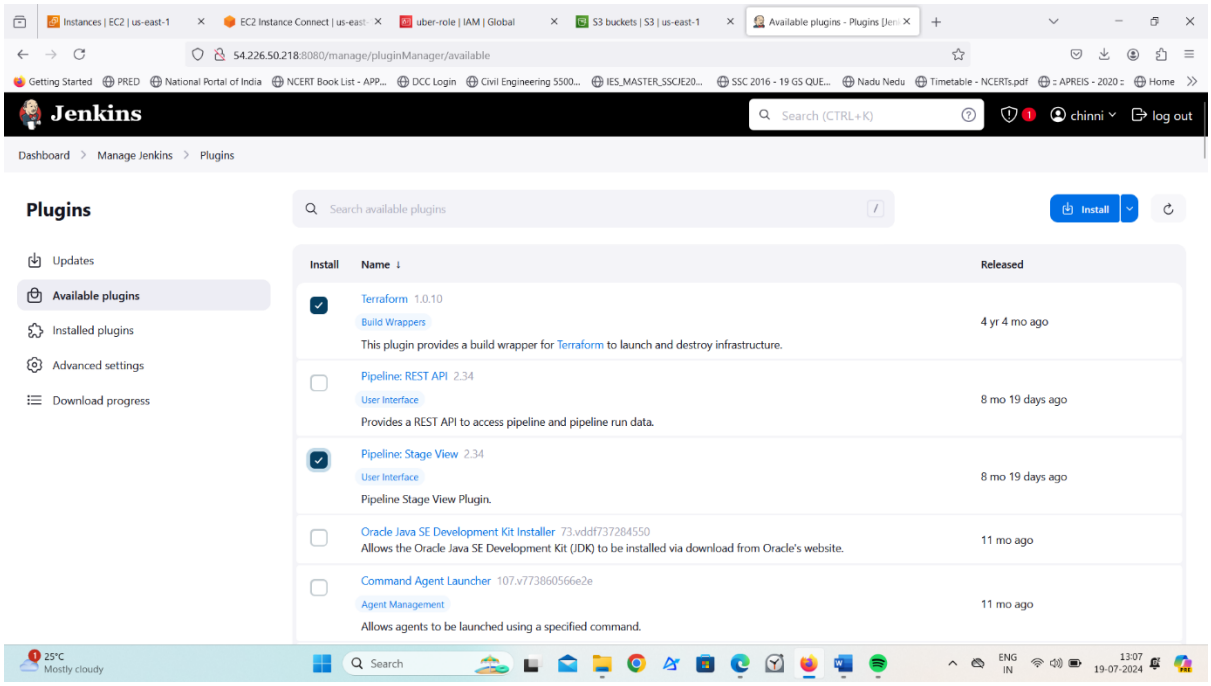
Go to Jenkins dashboard → Manage Jenkins → Plugins

Available Plugins, Search for Terraform and install it.





I also installed pipeline stage view to see our stages.

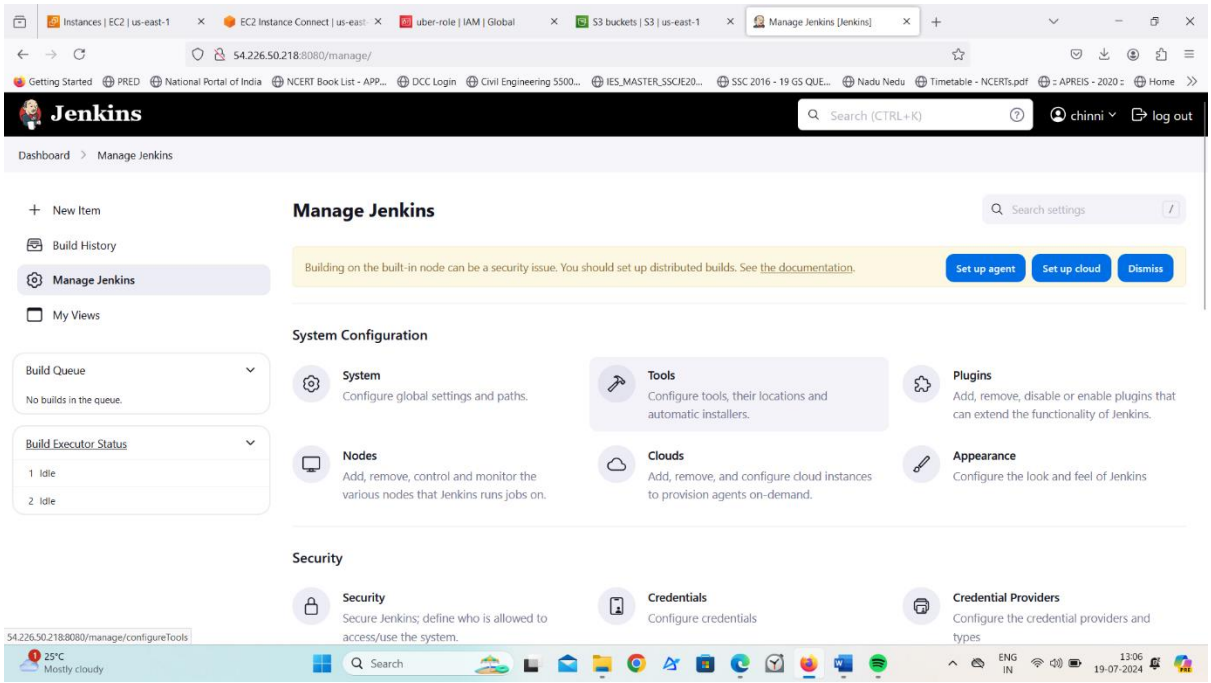


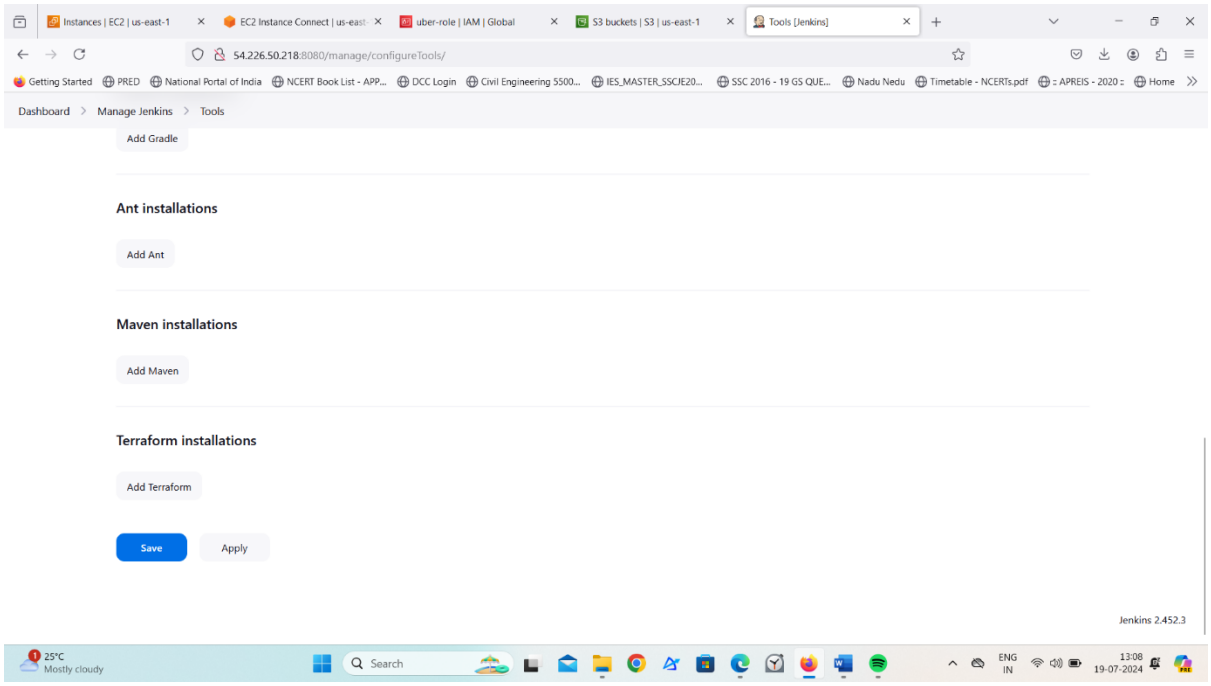
let’s find the path to our Terraform (we will use it in the tools section of Terraform)

which terraform

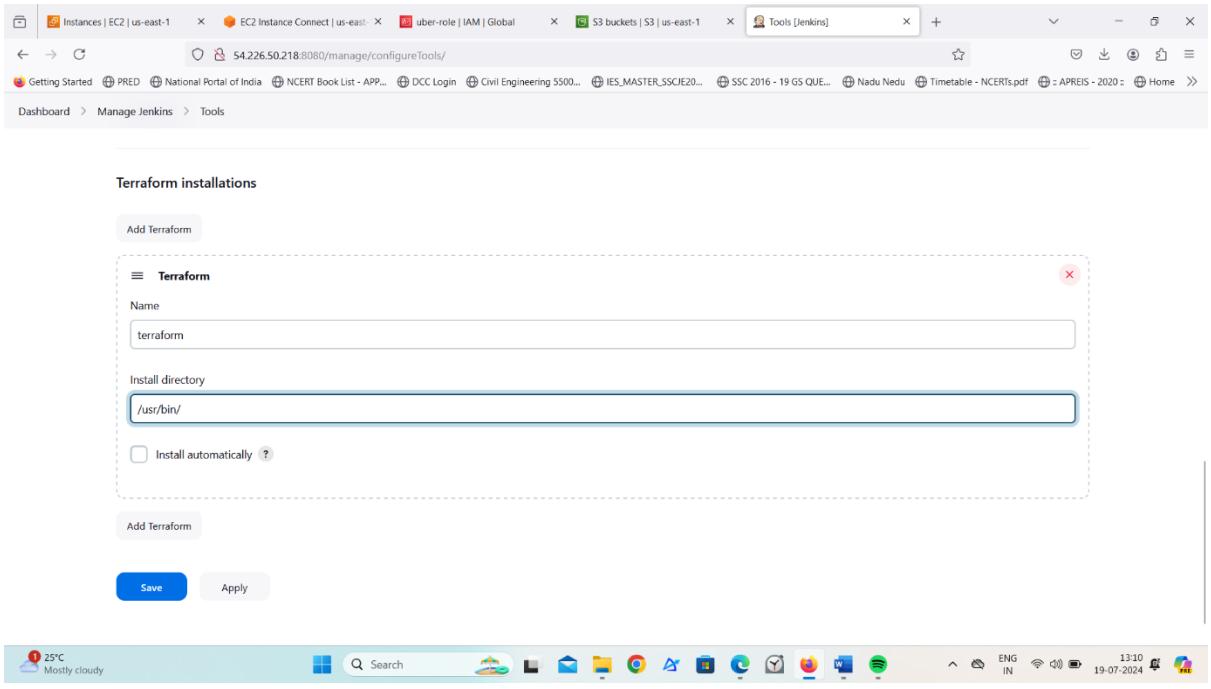
Now come back to Manage Jenkins → Tools

Add the terraform in Tools





Give name for it and give install directory (where we installed terraform)

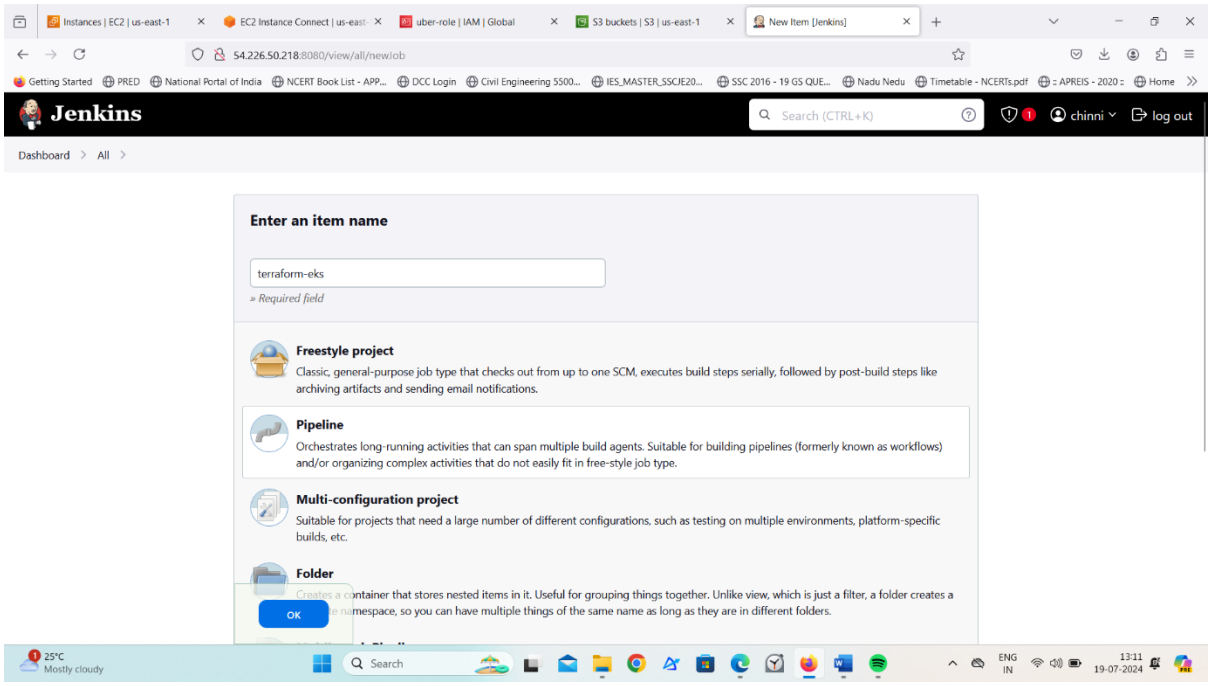


Click on apply and then save it

Now go to Jenkins dashboard.

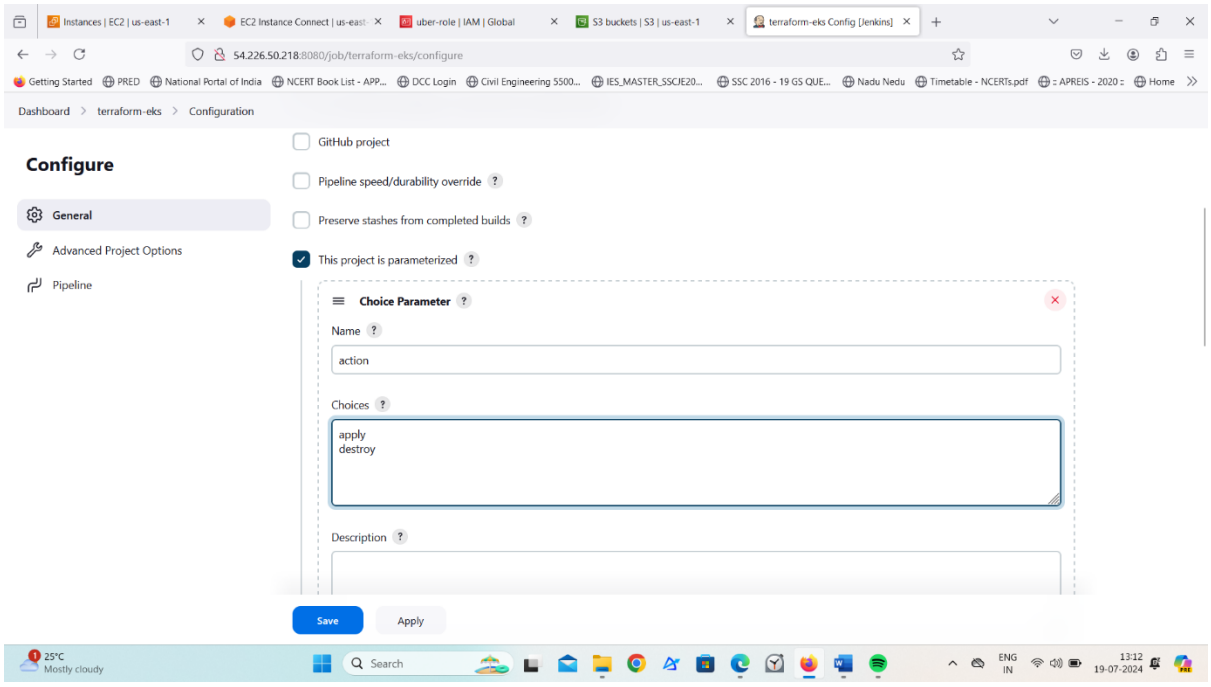
Create a new job for the EKS provision

Click on new item.



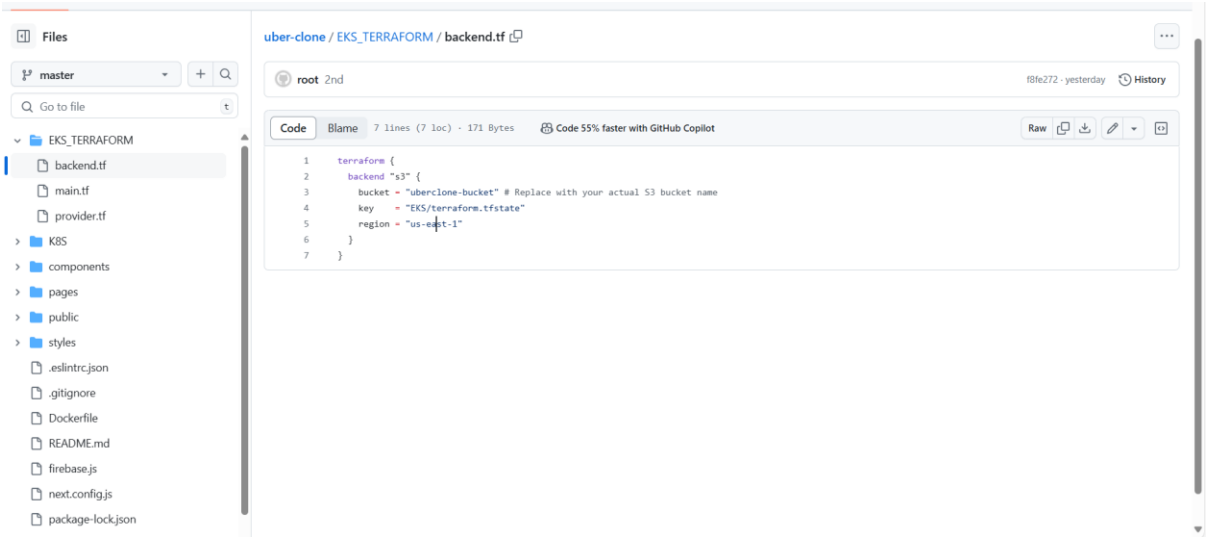
I want to do this with build parameters to apply and destroy while building only.

you have to add this inside job like the below image

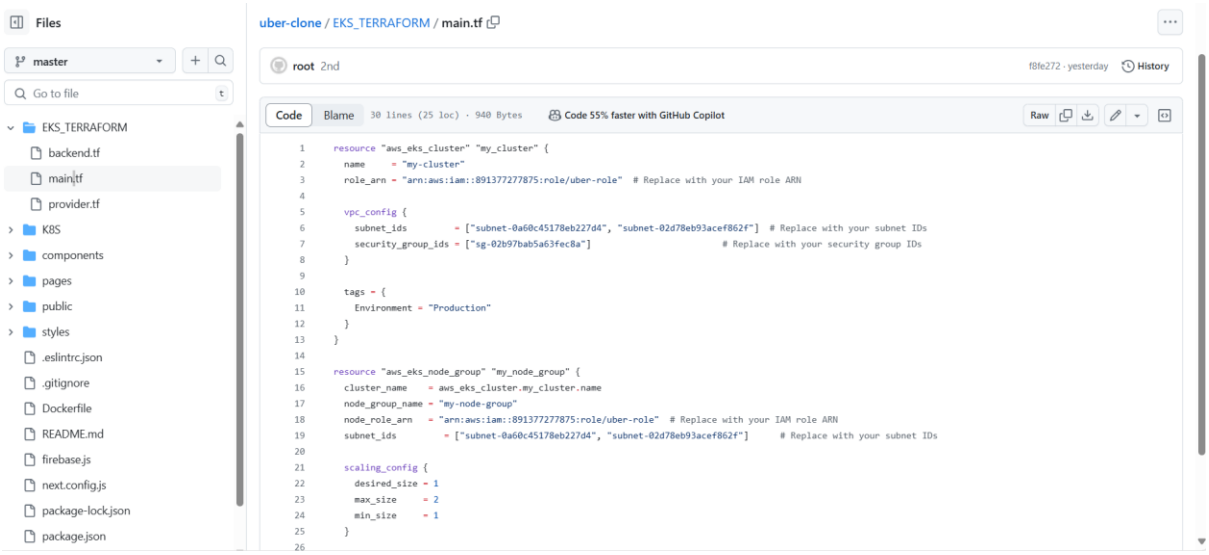


Before adding pipeline script, we have to do some changes in terraform configuration file.

Go to git hub and change bucket name in backend.tf file.



Also change role_arn, subnet ids, security group id in main.tf file.

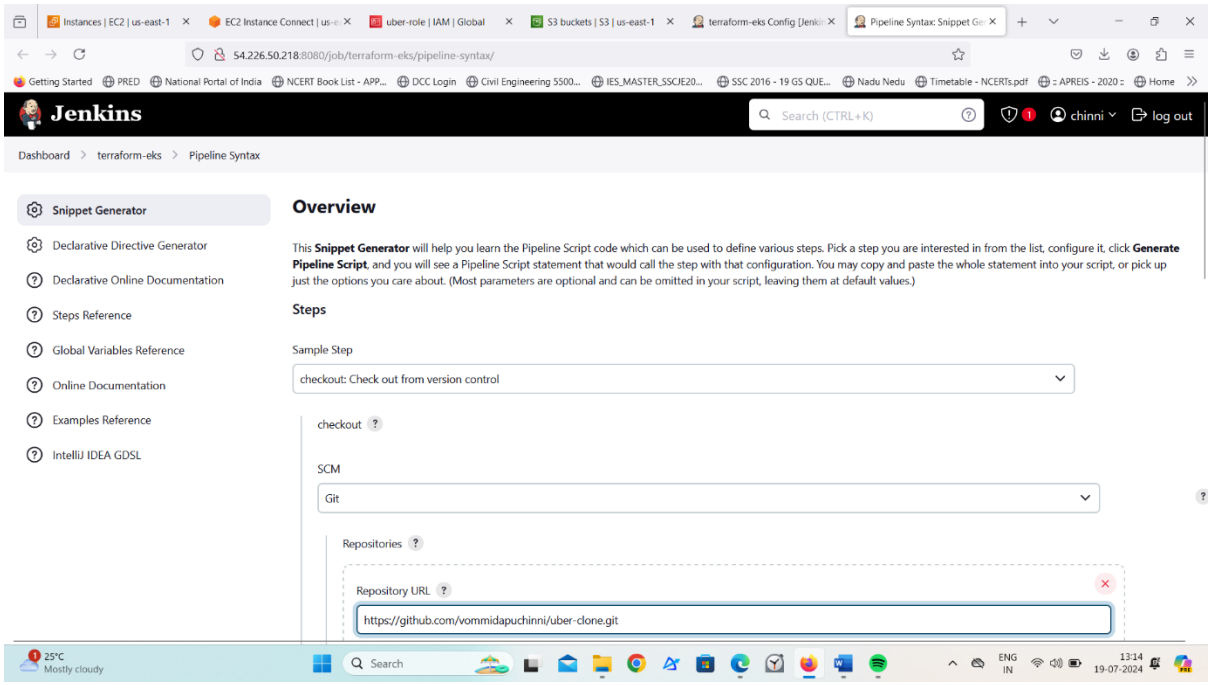


In provider.tf change your region.

```
pipeline {
    agent any
    stages {
        stage ('Checkout from Git') {
            steps {
                checkout scmGit(branches: [[name: '*/master']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/vommidapuchinni/uber-clone.git']])
            }
```

```
}
stage ('Terraform version'){
    steps{
        sh 'terraform --version'
    }
}
stage('Terraform init'){
    steps{
        dir('EKS_TERRAFORM') {
            sh 'terraform init'
         }
    }
}
stage('Terraform validate'){
    steps{
        dir('EKS_TERRAFORM') {
            sh 'terraform validate'
         }
    }
}
stage('Terraform plan'){
    steps{
        dir('EKS_TERRAFORM') {
            sh 'terraform plan'
         }
    }
}
```

```
        stage('Terraform apply/destroy'){
            steps{
                dir('EKS_TERRAFORM') {
                    sh 'terraform ${action} --auto-approve'
                }
            }
        }
    }
}
```



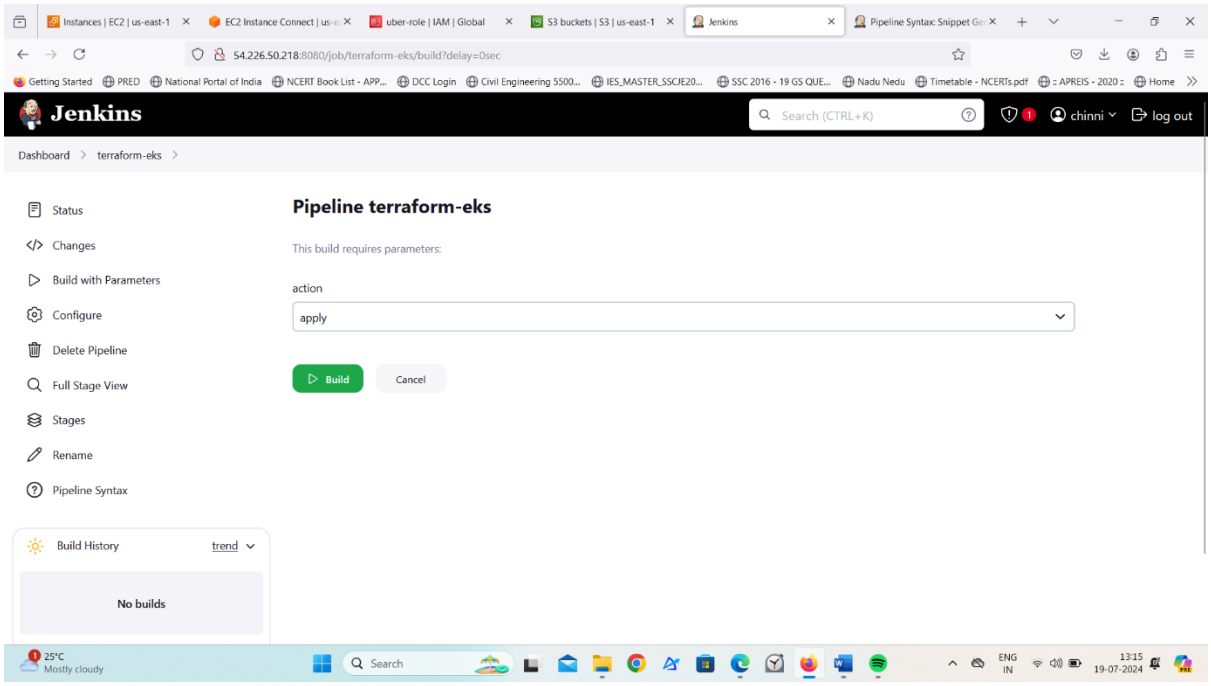
From this pipeline syntax we get our checkout syntax.

Click on apply and then save.

Click on build parameters.

It will ask for actions to apply/destroy.

Choose apply and click on build.

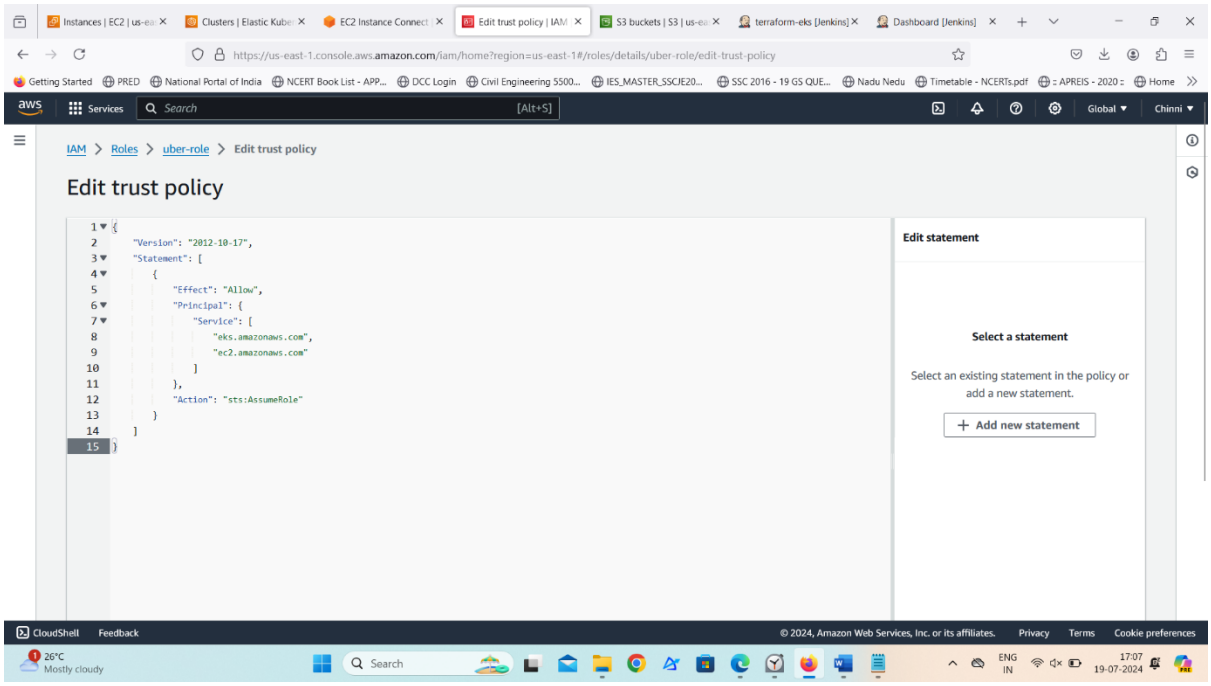


It will take few minutes to create it.

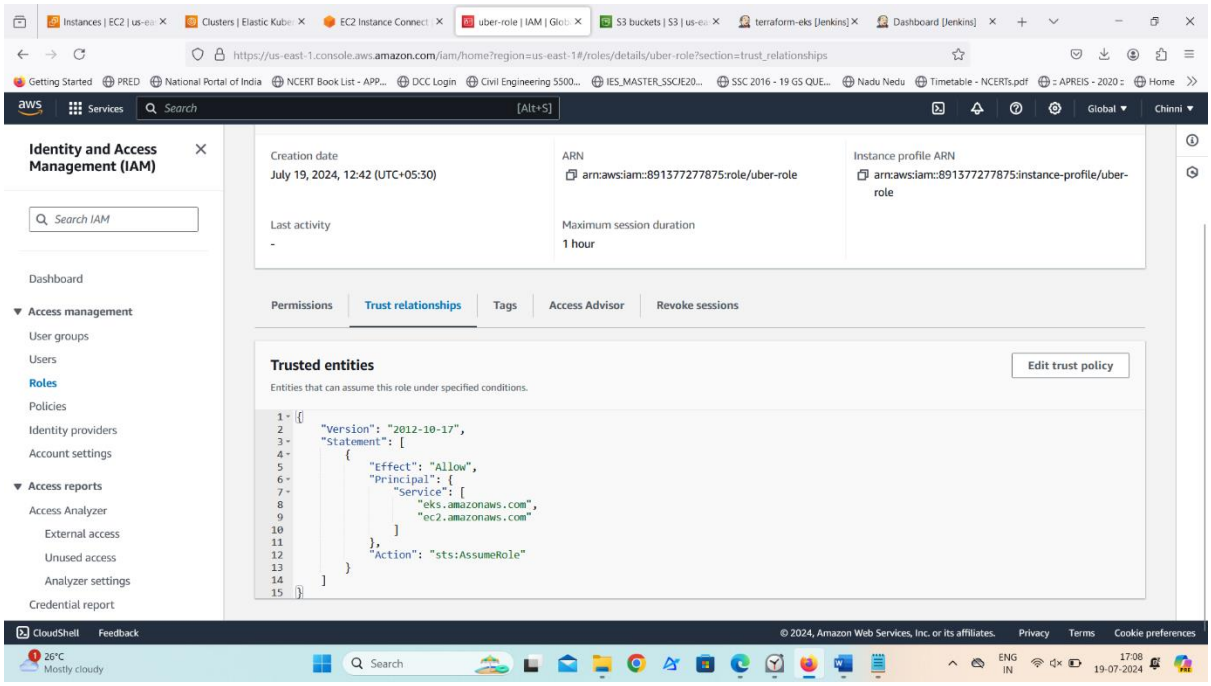
First, I got error to create it because I didn't give permission to create node group.

Now go to its IAM role, under trust relationships, click on edit trust policy.

Give eks permission along with ec2.

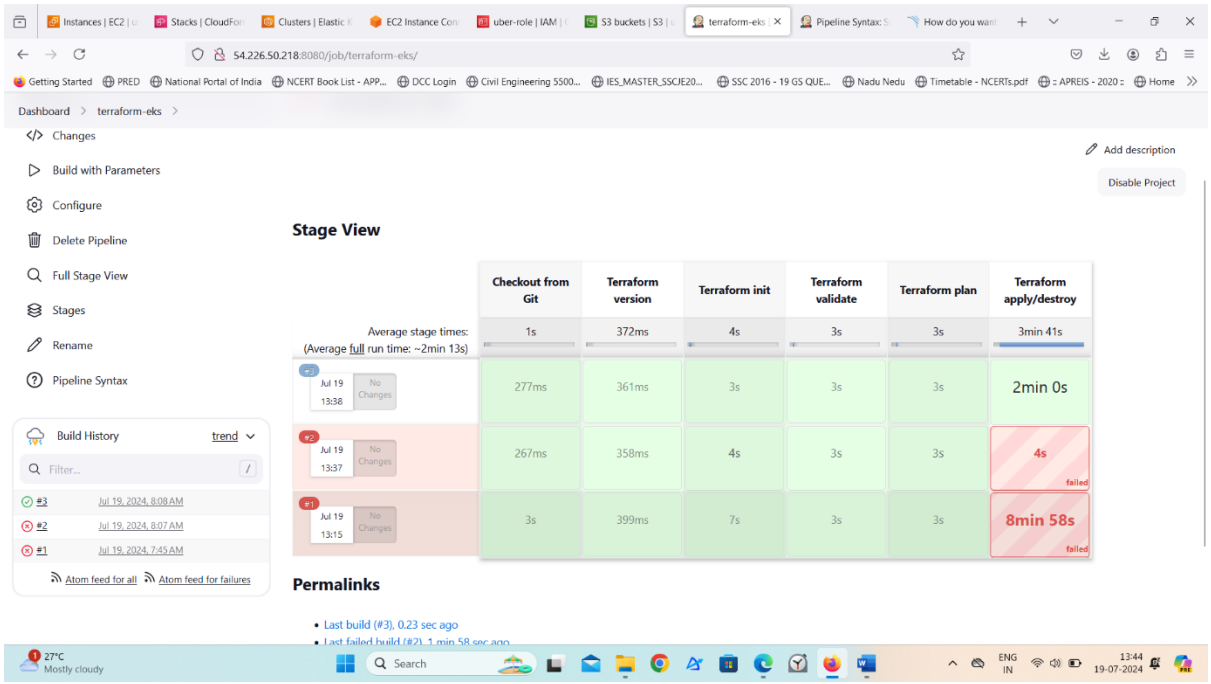


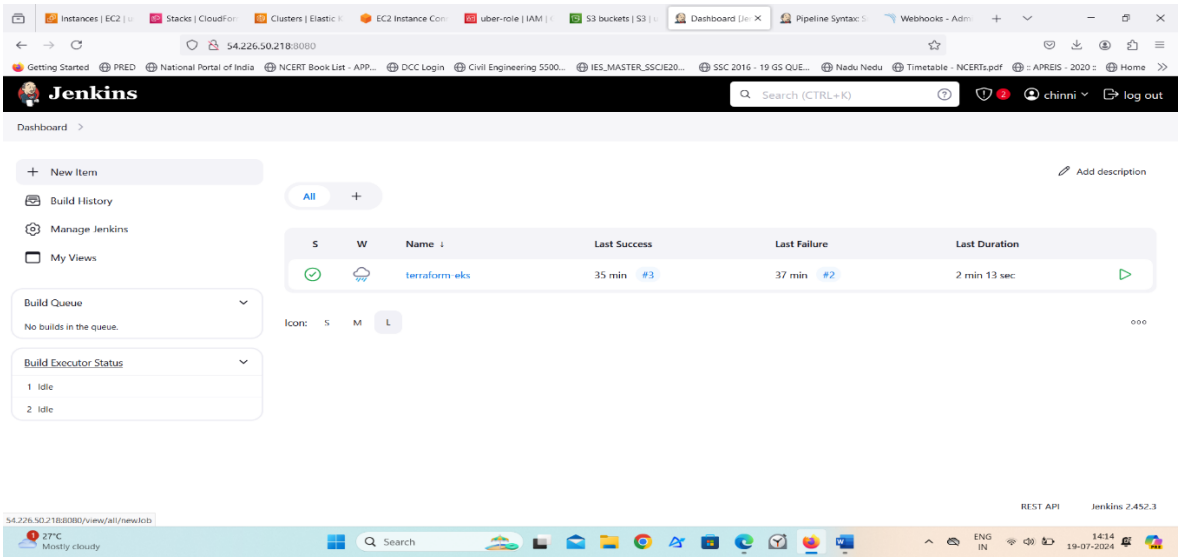
Click on save policy.



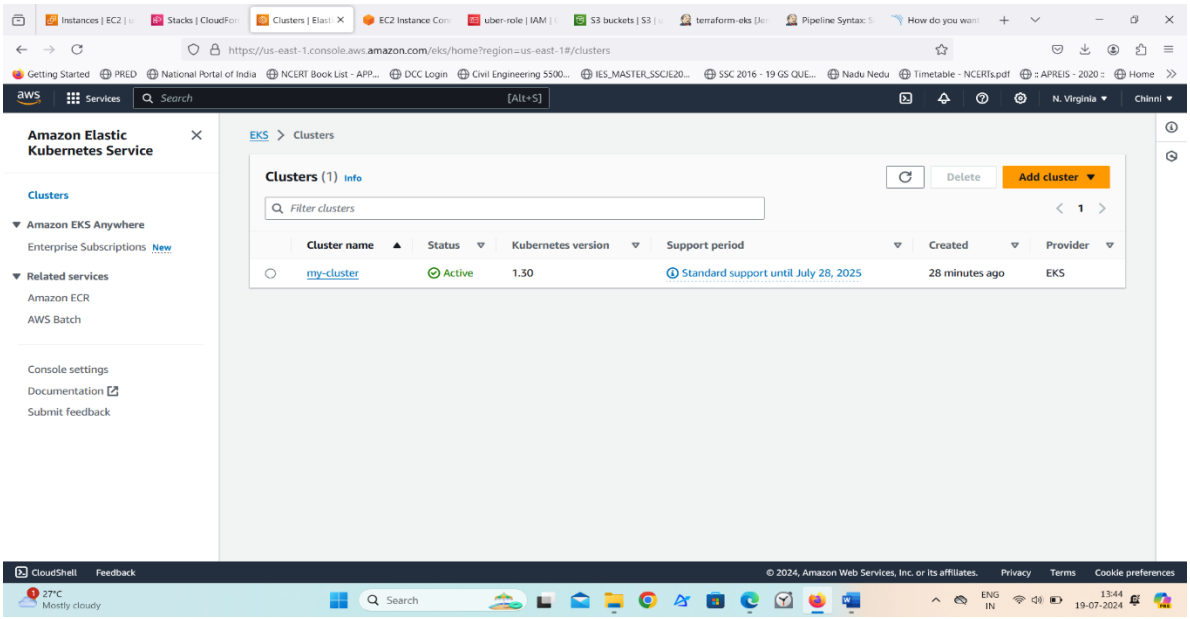
We see like this under IAM role.

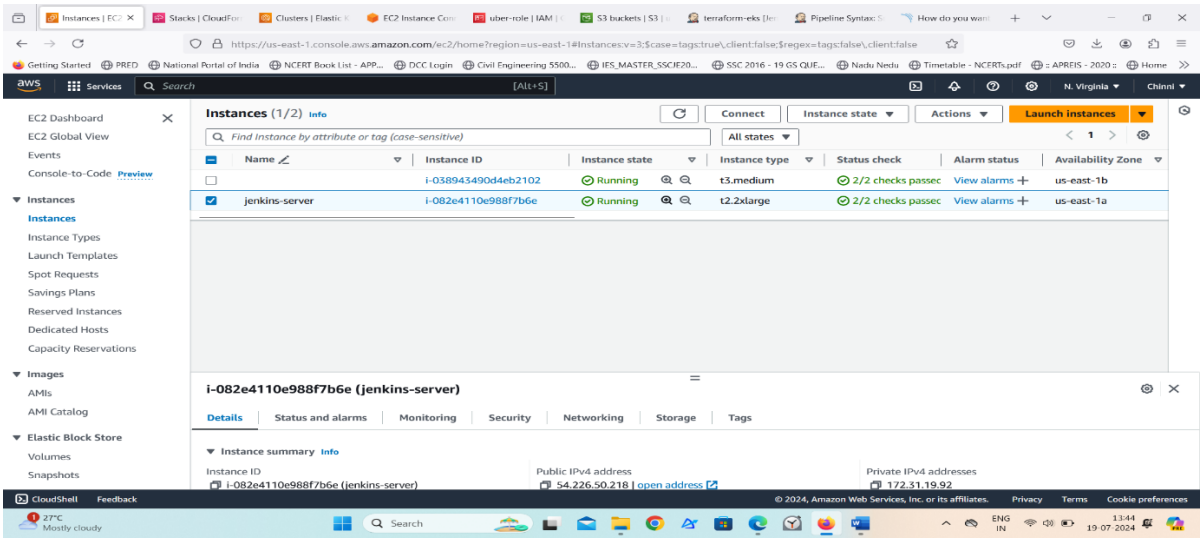
After editing it come to Jenkins server again click on build with parameters choose apply and click on build.





Go to AWS console we see our cluster is created and one node group also.





Step7: Plugins installation & setup (Java, Sonar, Nodejs, OWASP, Docker)

Go to Jenkins dashboard

Manage Jenkins → Plugins → Available Plugins

Search for the Below Plugins

Eclipse Temurin installer

SonarQube Scanner

NodeJS

OWASP Dependency-Check

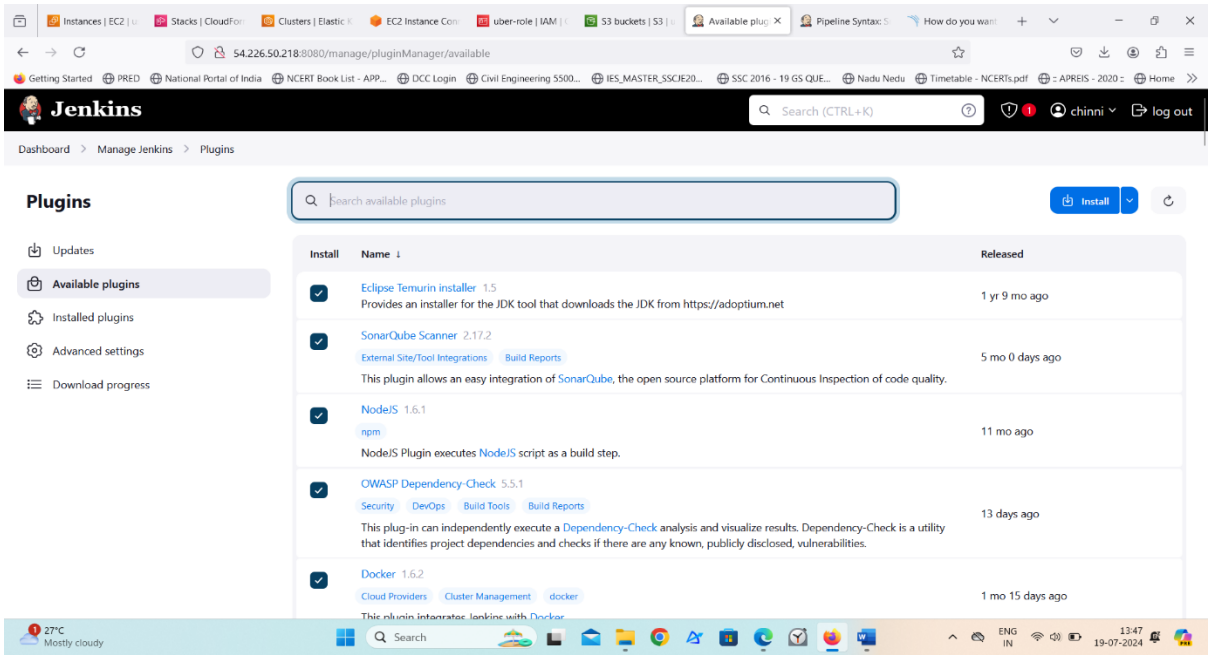
Docker

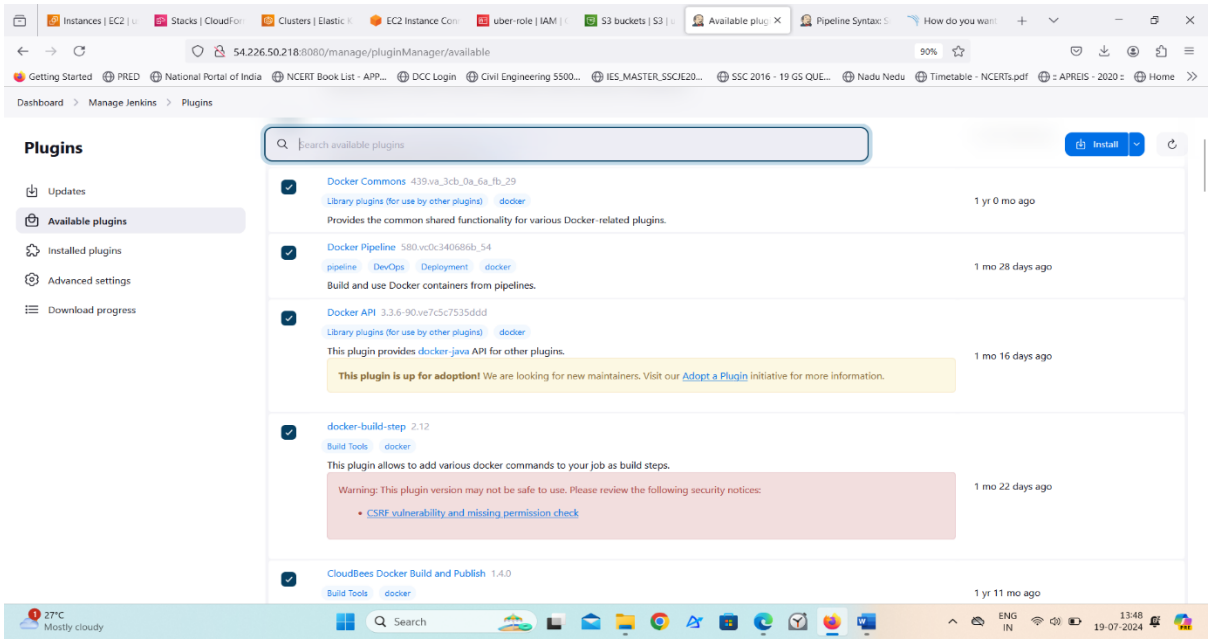
Docker Commons

Docker Pipeline

Docker API

Docker-build-step



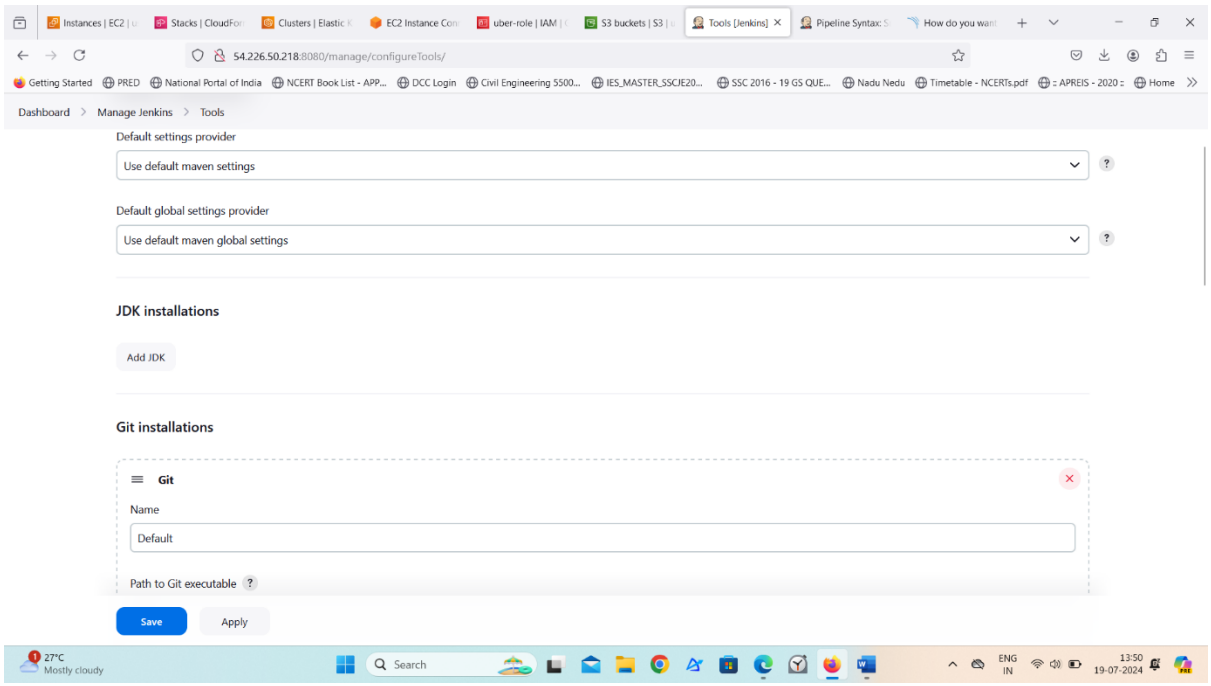


Step8: Configure in Global Tool Configuration

Goto Manage Jenkins → Tools

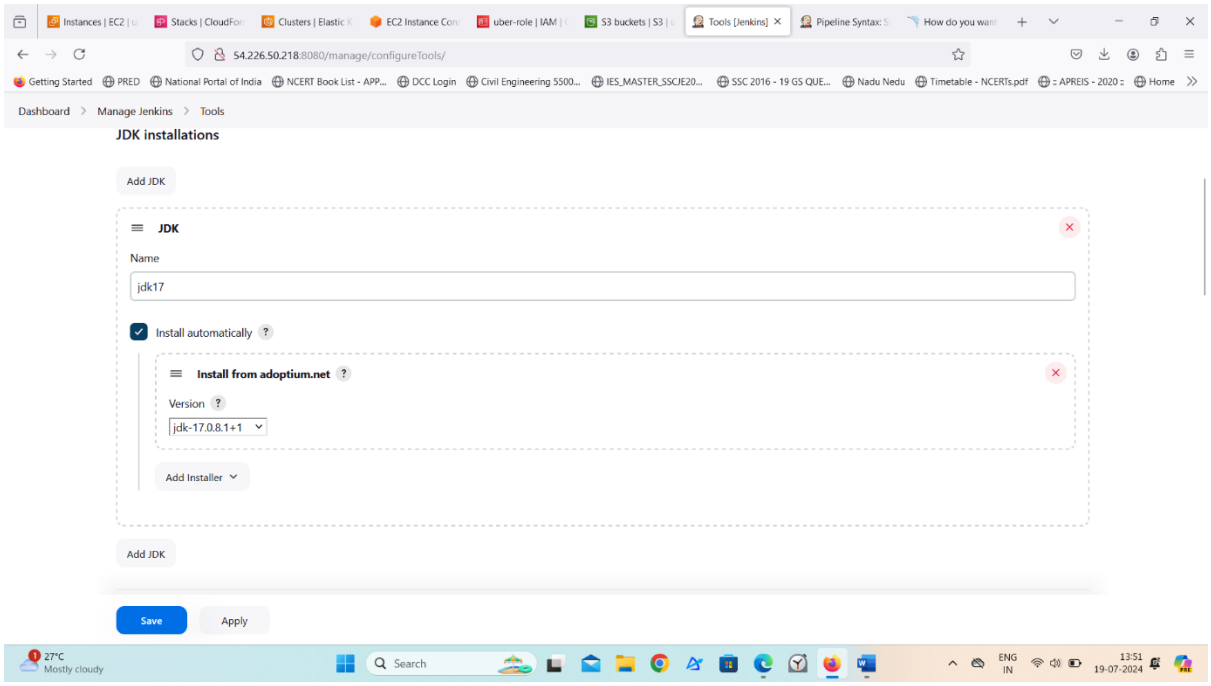
Search for JDK installation.

Click on add JDK.



Give name and click on install automatically. Click on add installer.

Choose install from adoptium.net, choose version.

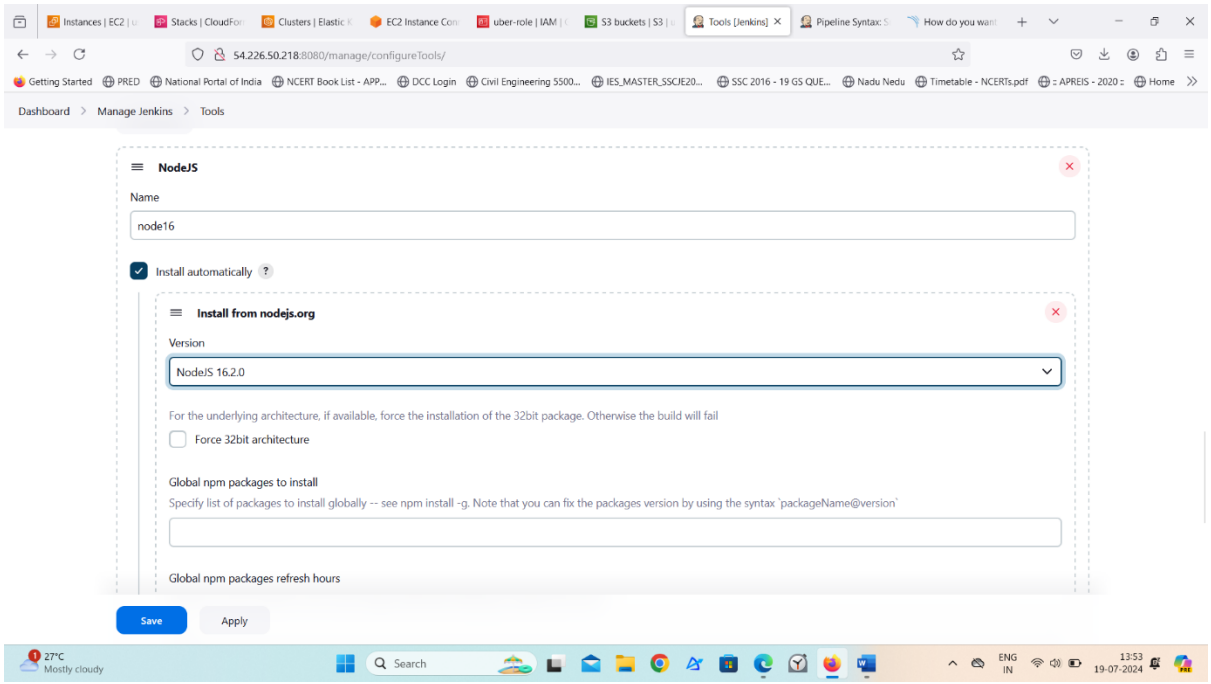


Come down search NodeJS.

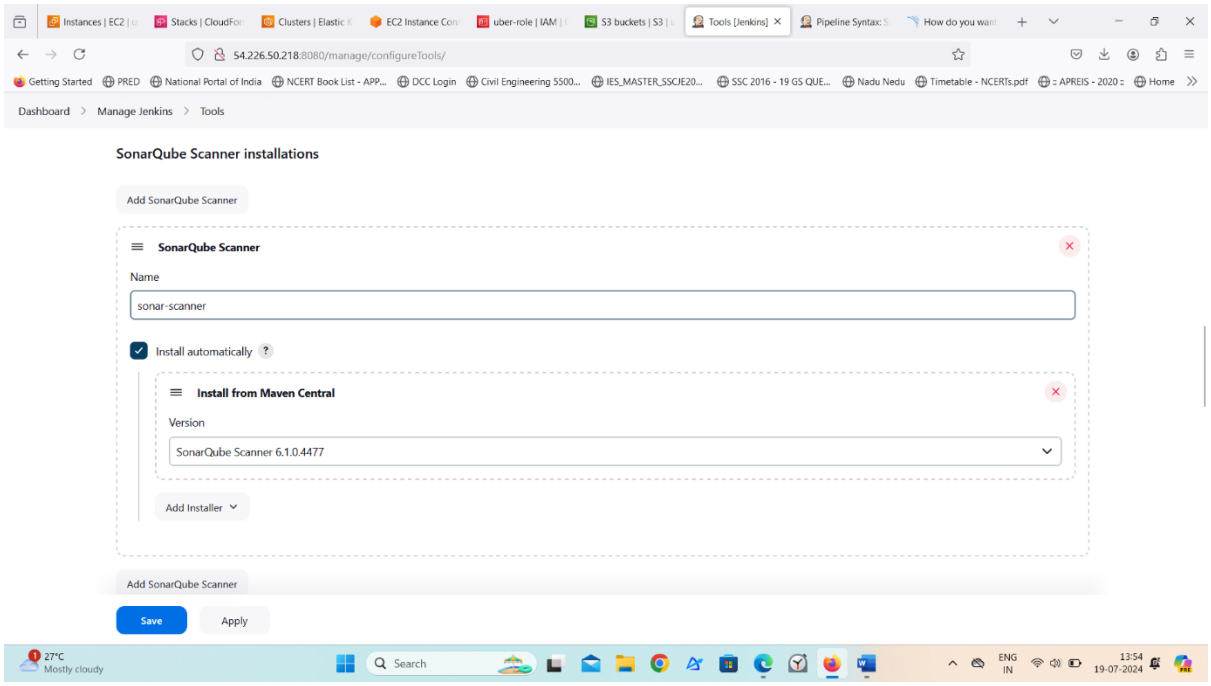
Give name and click on install automatically.

Click on add installer, choose install from nodejs.org.

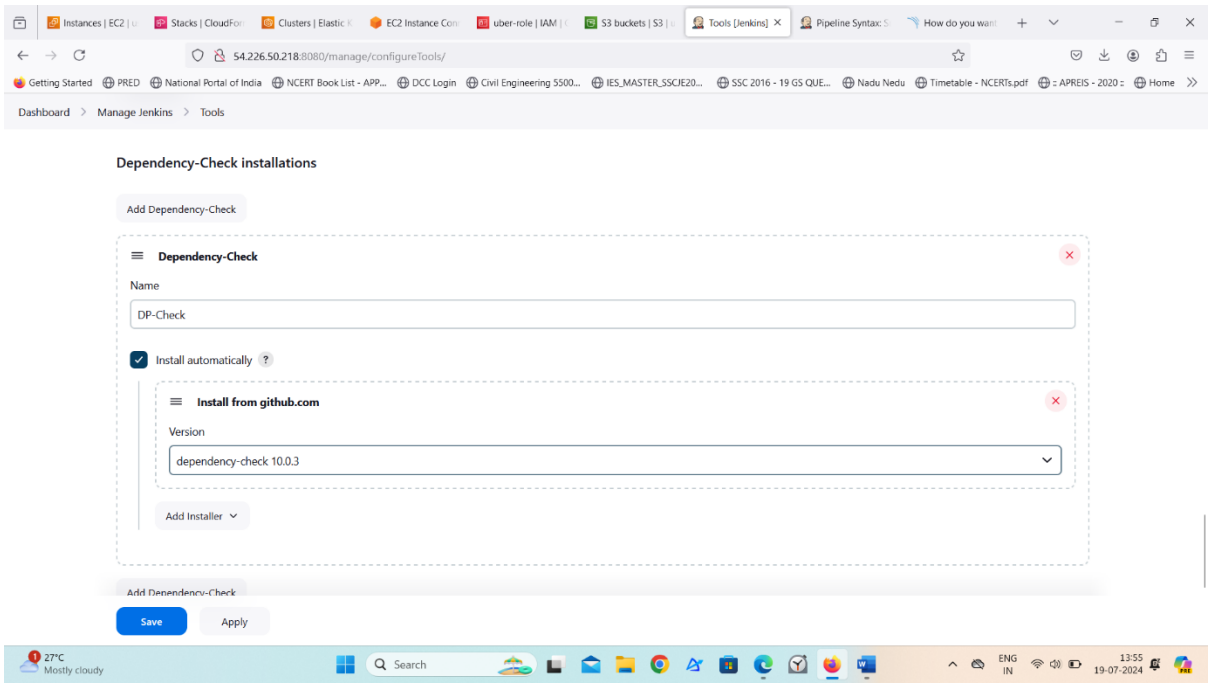
Choose version.



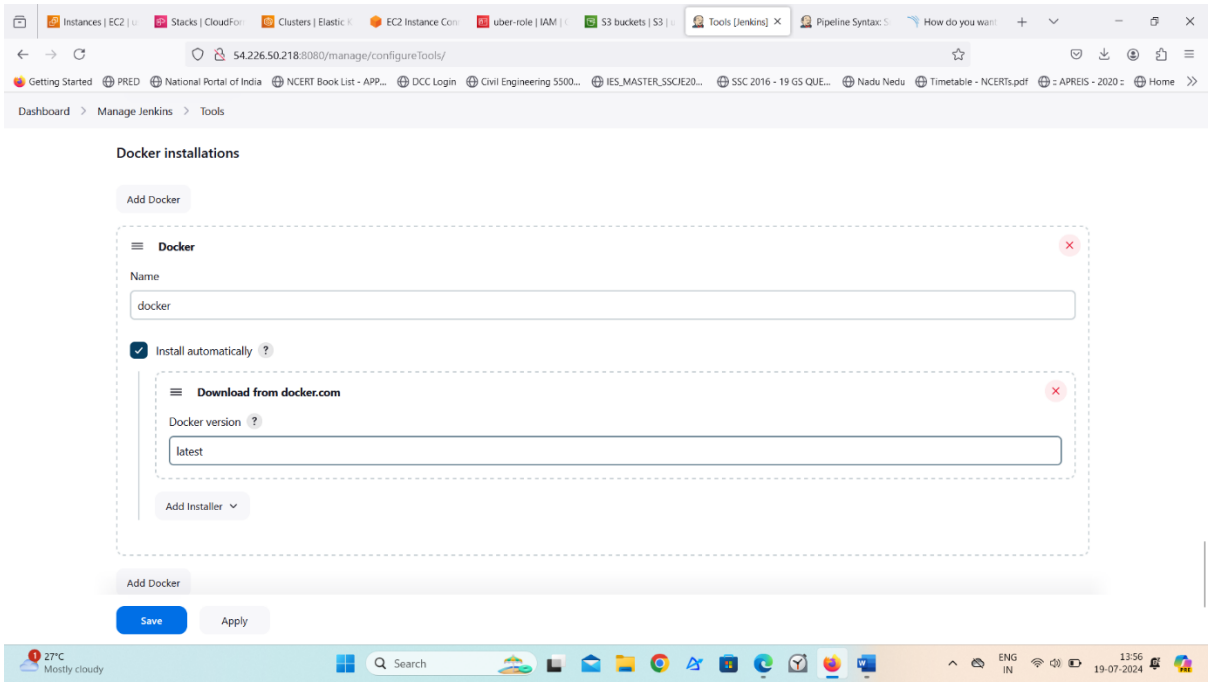
Search for SonarQube scanner installation, give name and click on install automatically. Click on add installer choose install from maven central.



- Come down search dependency check installation. Click on add dependency check.
- Give name and click on install automatically, click on add installer.
- Choose install from github.com.



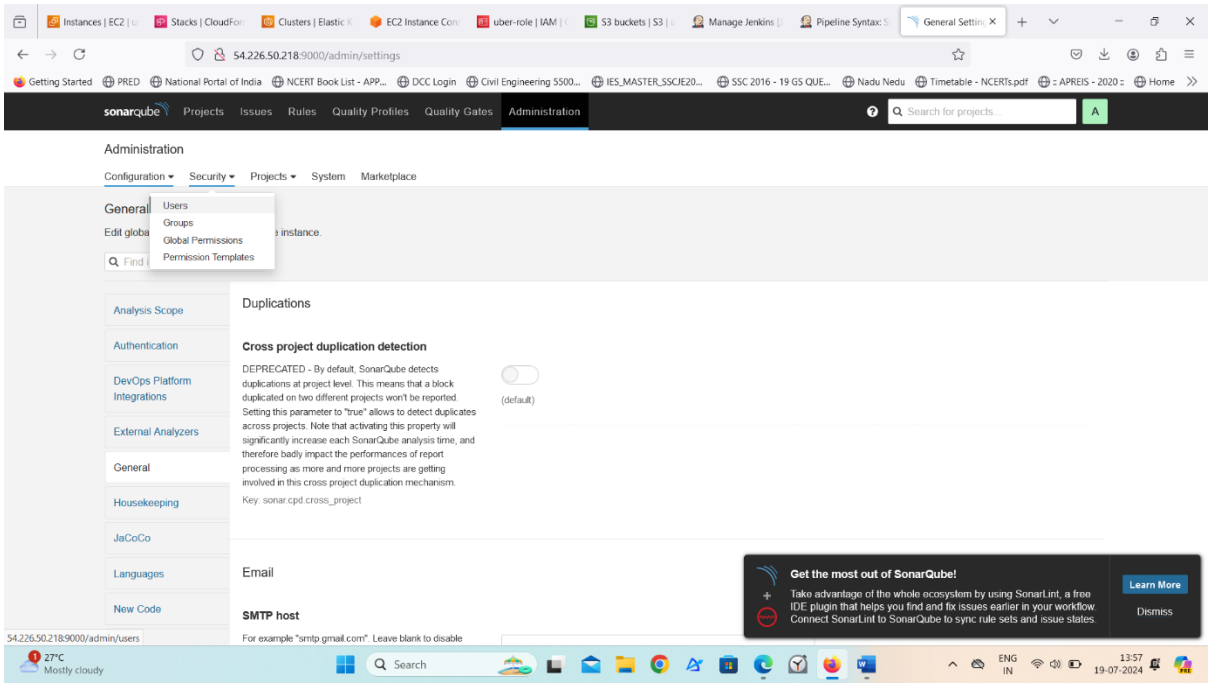
- Come down search for docker installation.
- Click on add docker. Give name. click on install automatically →add installer →choose download from docker.com →choose latest.



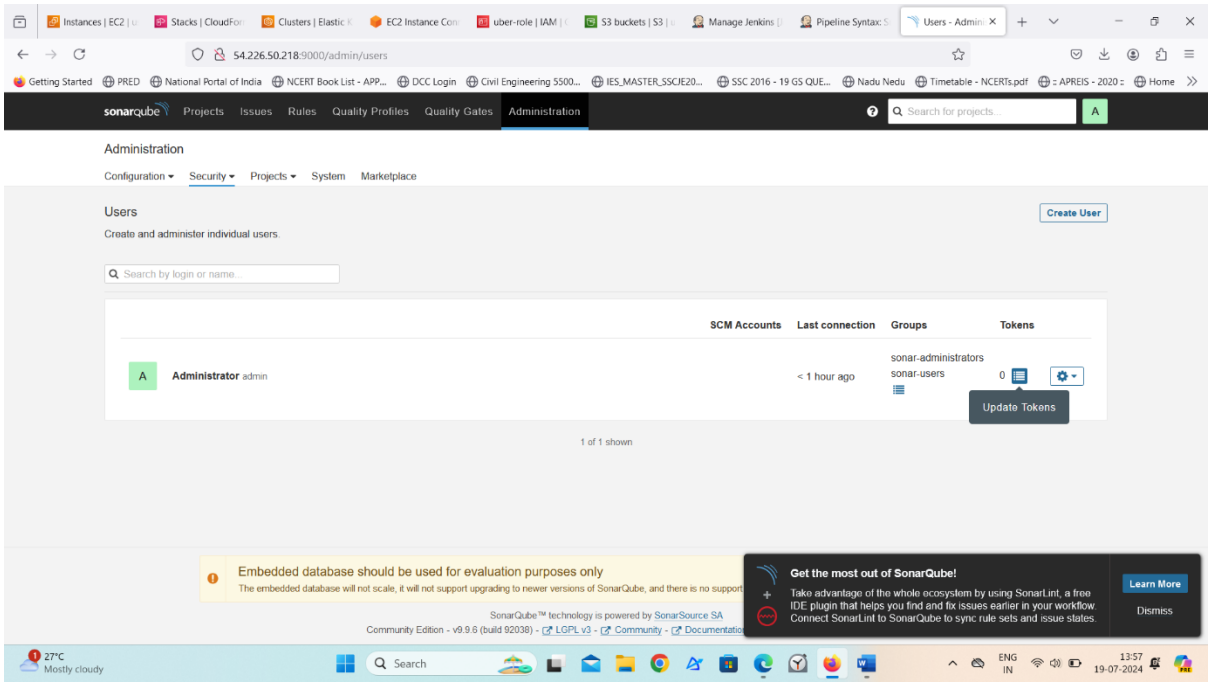
Click on apply and then save it.

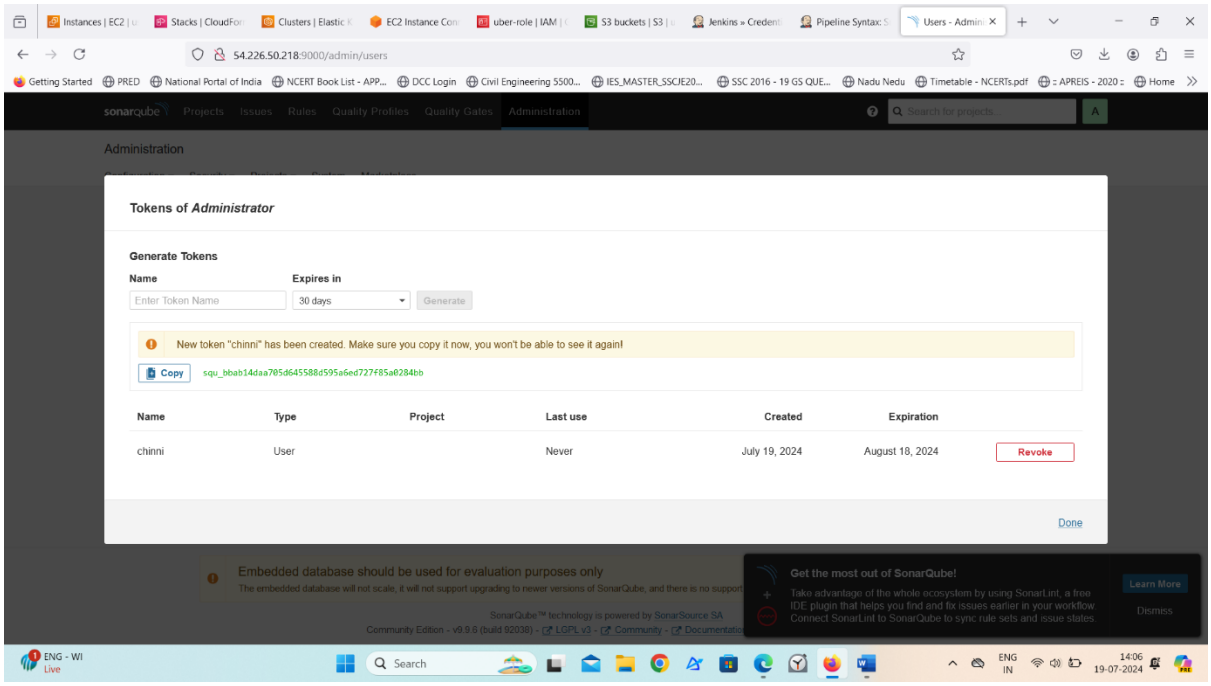
Step9: Configure Sonar Server in Manage Jenkins

Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token.



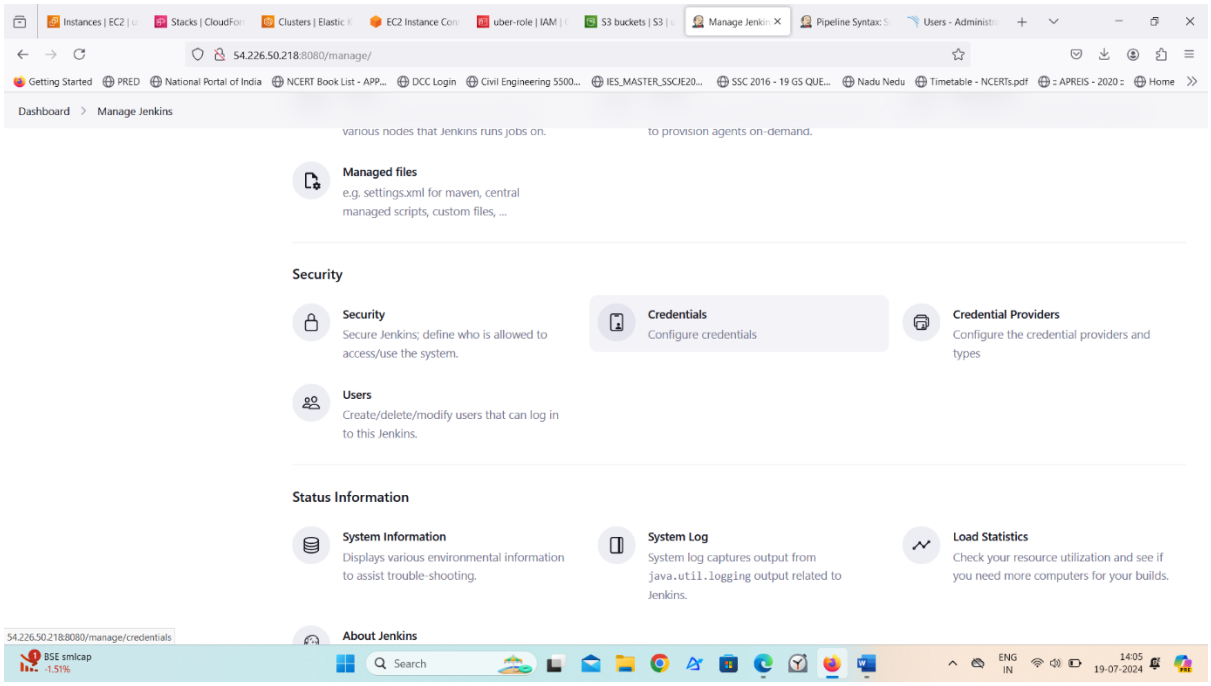
Create a token with a name and generate



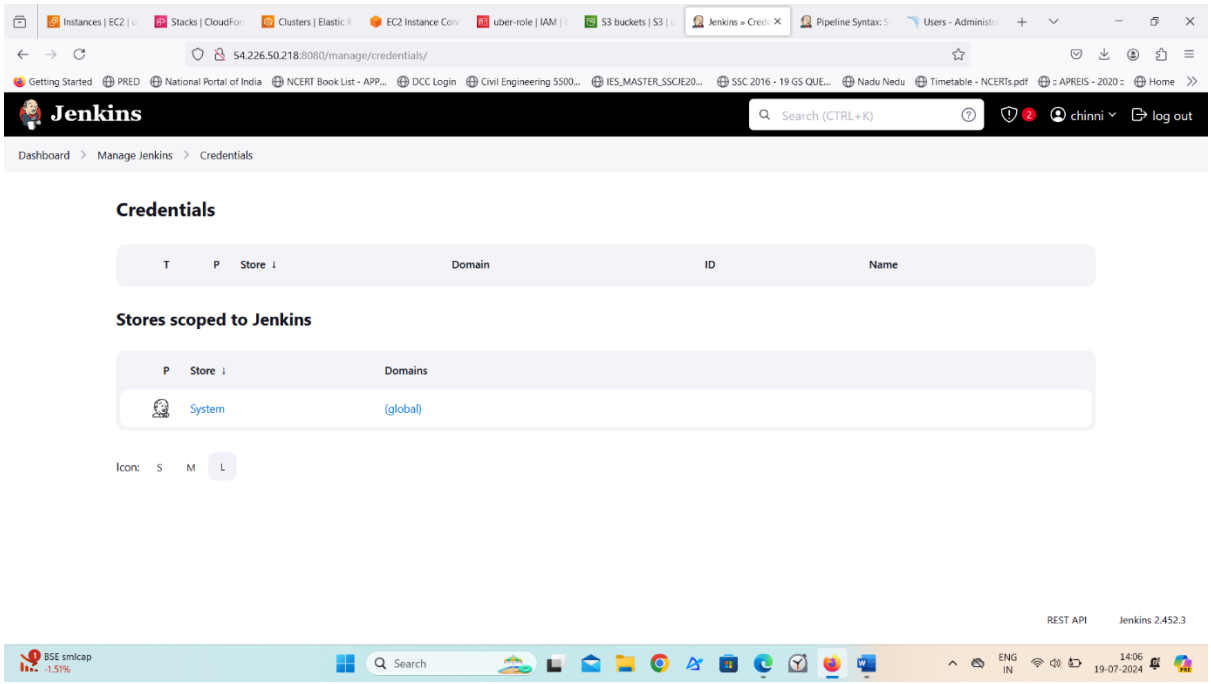


copy Token

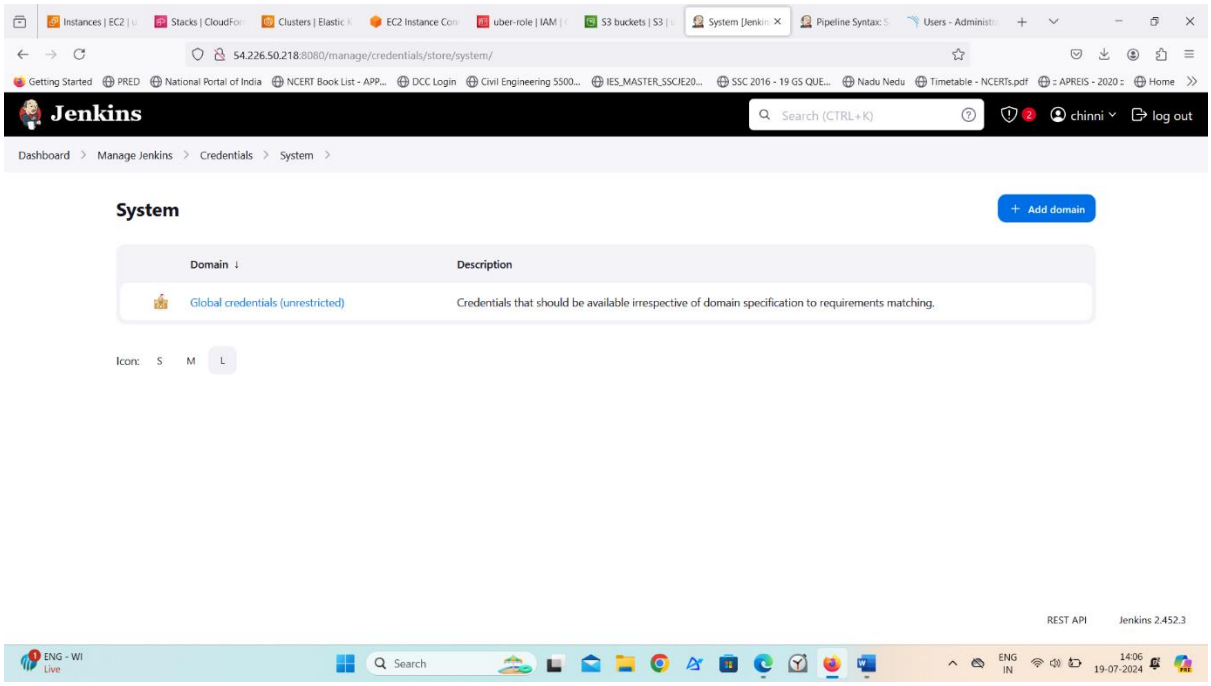
Goto Jenkins Dashboard → Manage Jenkins → Credentials.



Click on system.



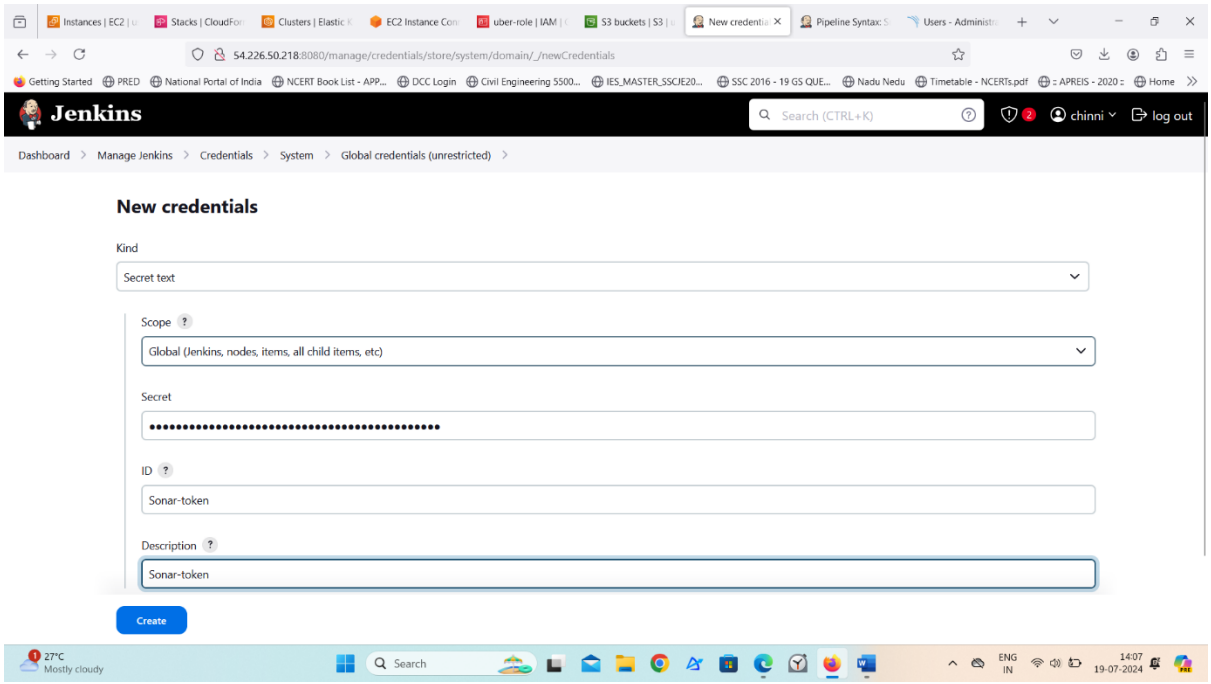
Click on global credentials (unrestricted).

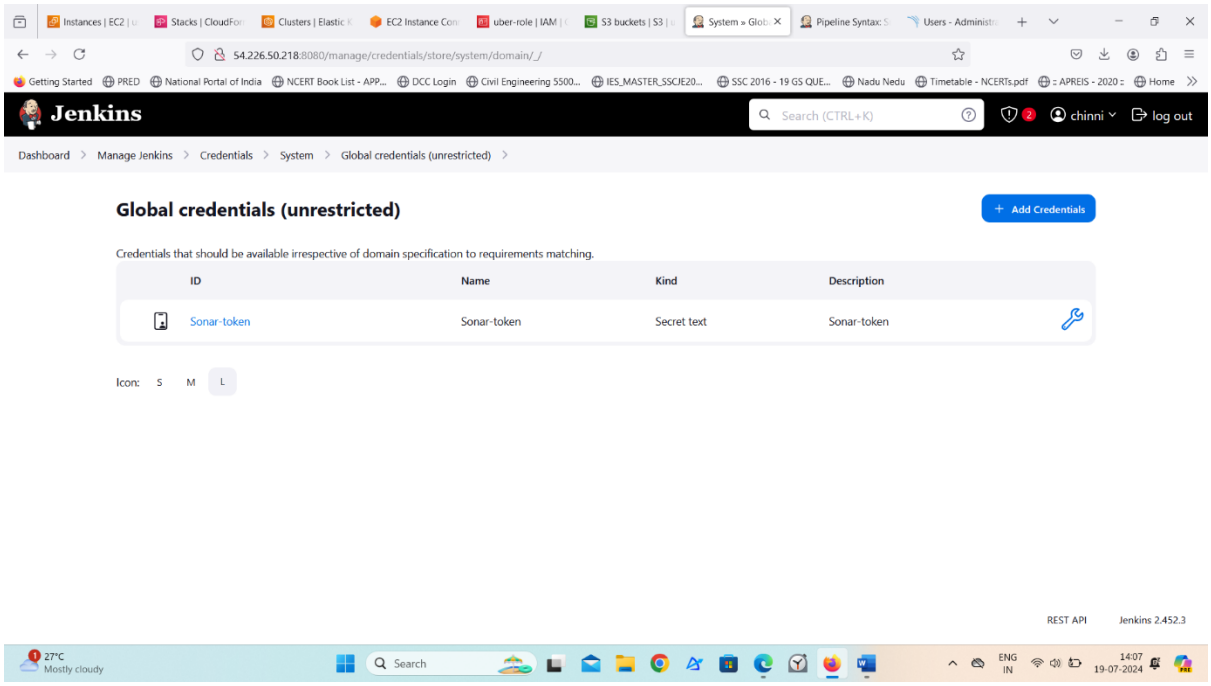


Choose kind as secret text.

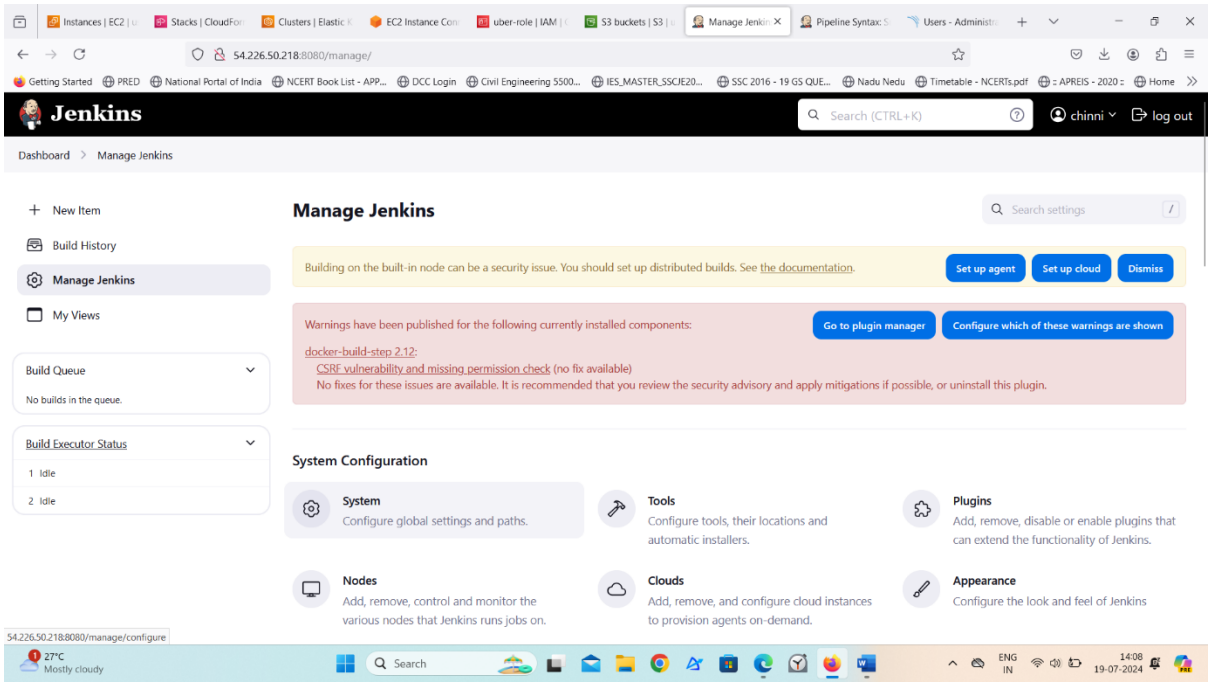
Paste the sonar token in secret and click on sonar-token.

Click on create.





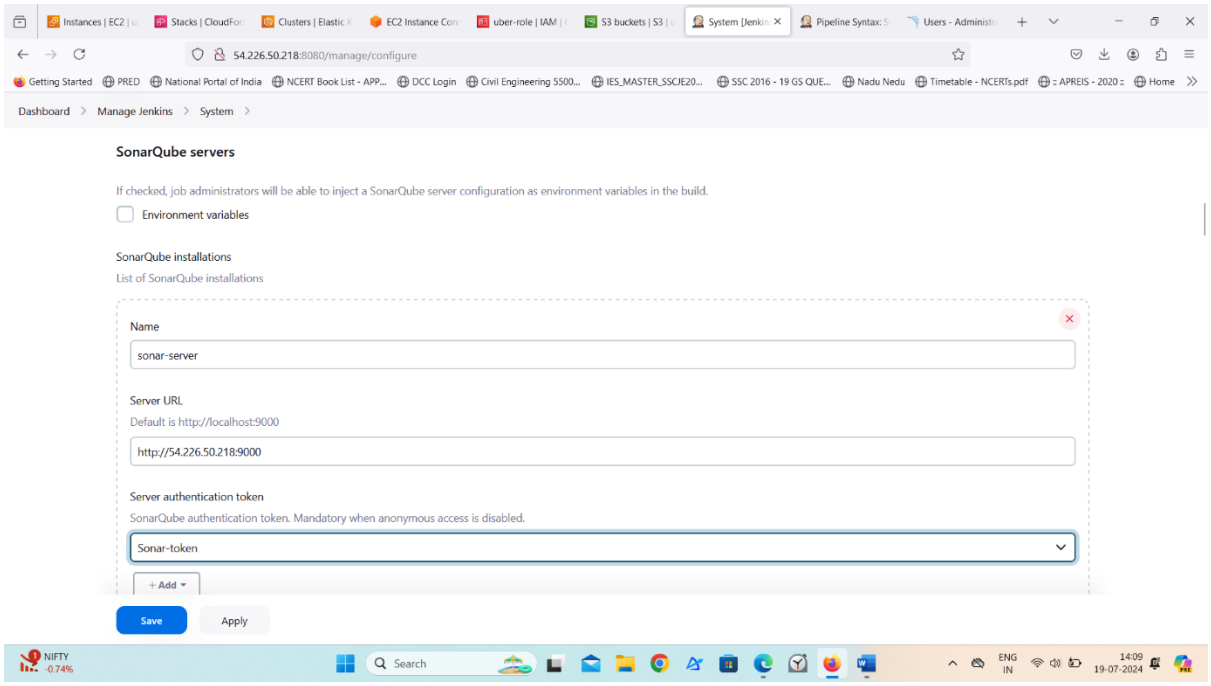
Now, go to Dashboard → Manage Jenkins → System.



Give name and sonar server URL.

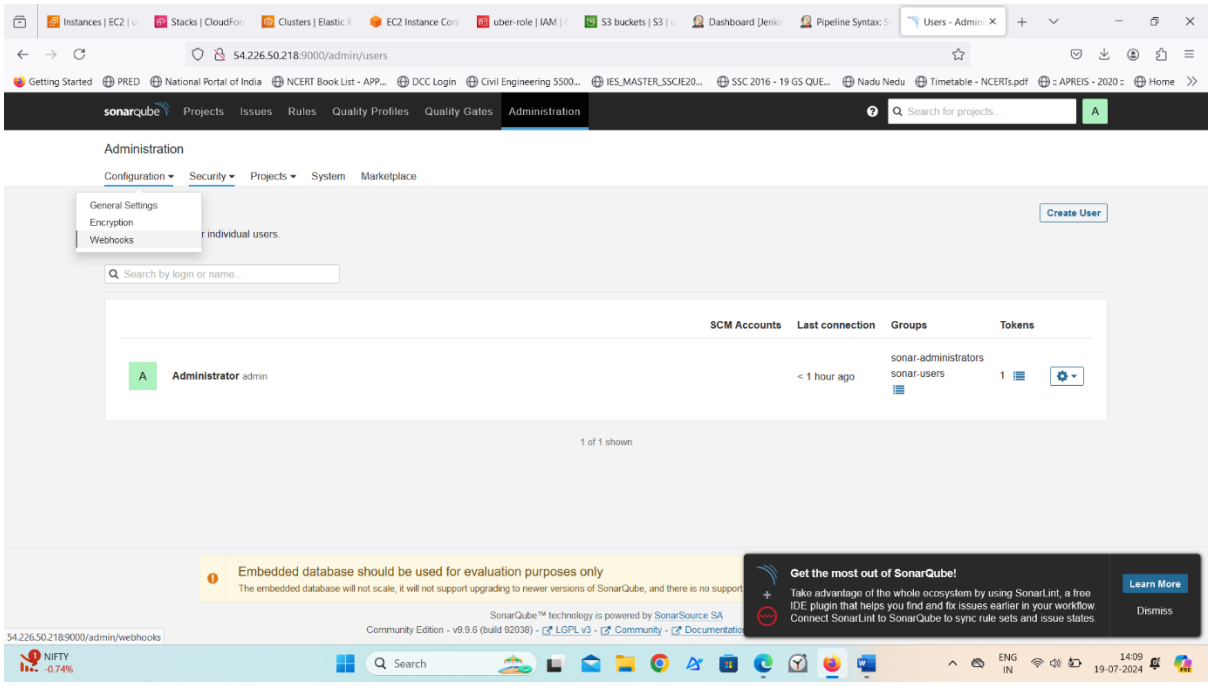
Select our token.

Click on apply and save

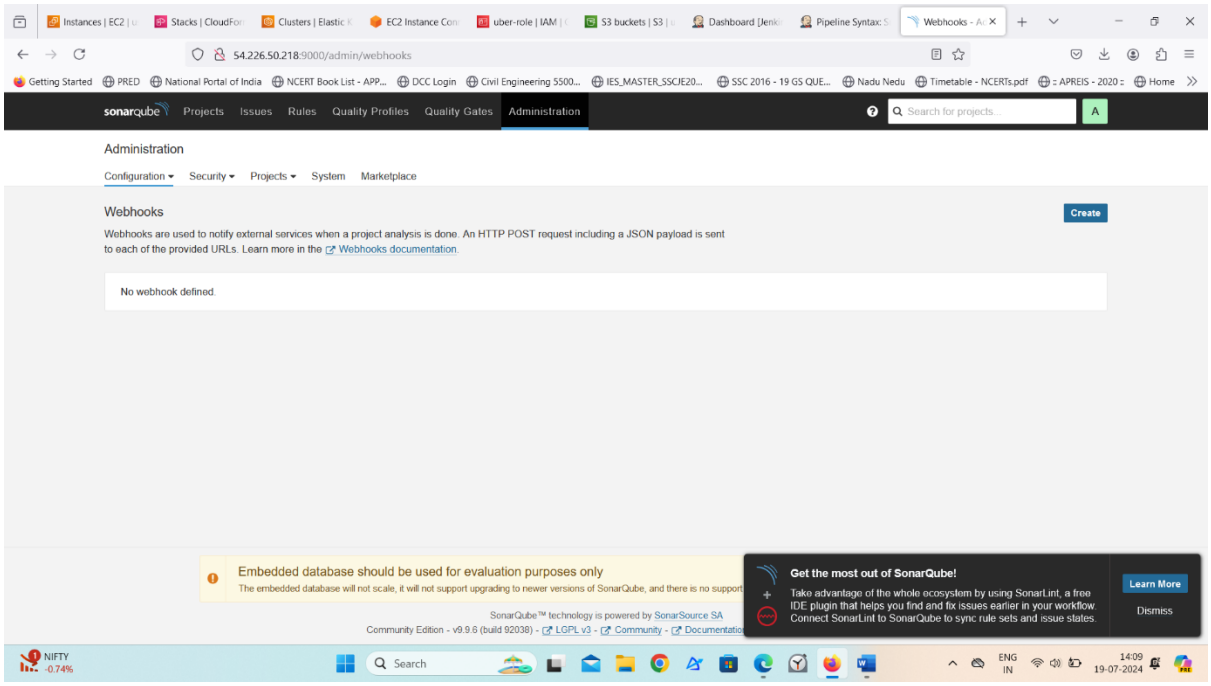


Go to SonarQube server.

Administration → Configuration →Webhooks



Click on create.

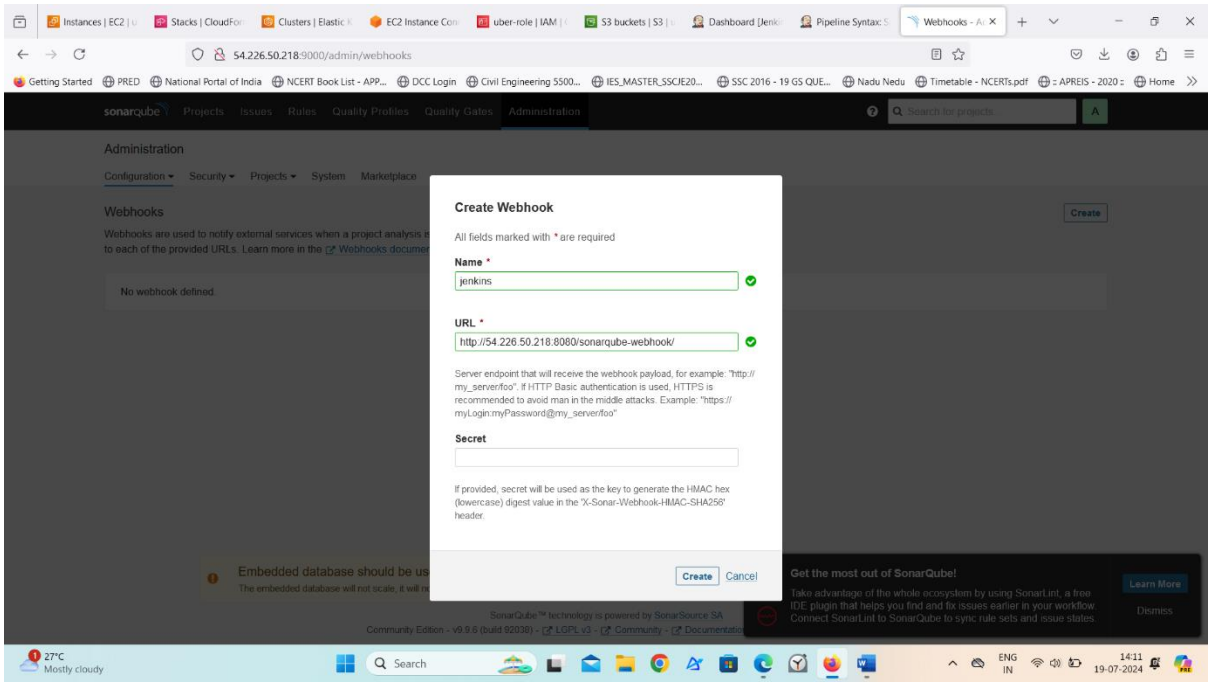


Add details

Give name

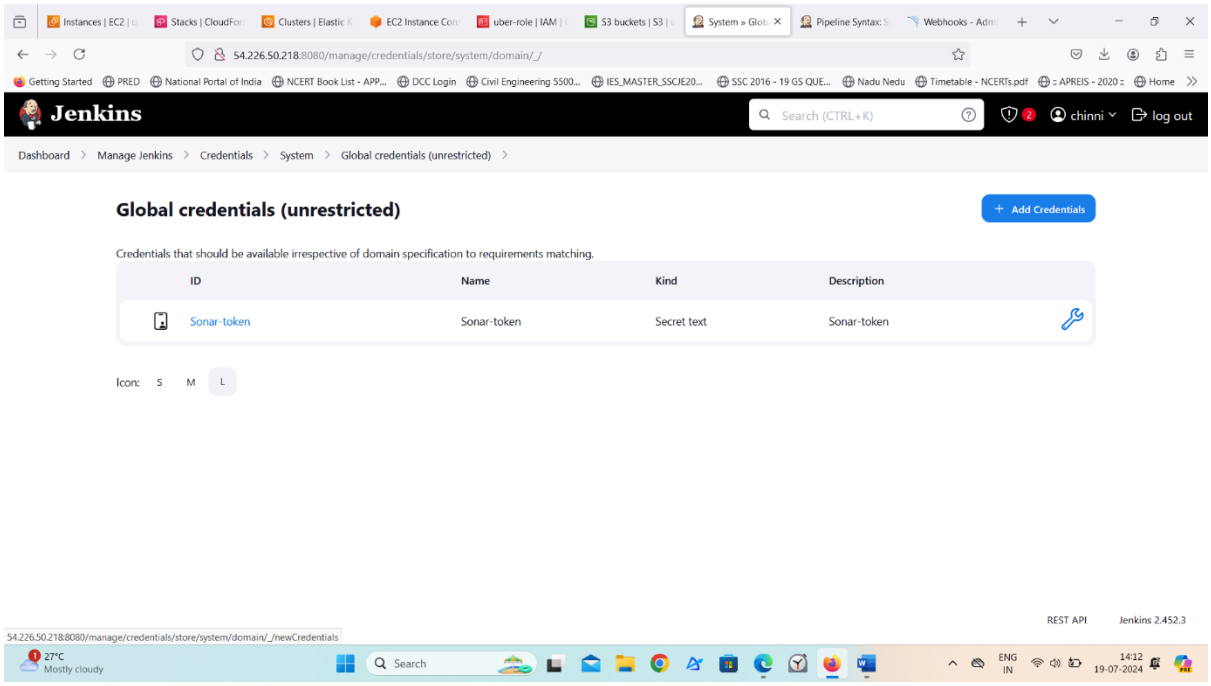
#in url section of quality gate

<http://jenkins-public-ip:8080>/sonarqube-webhook/>

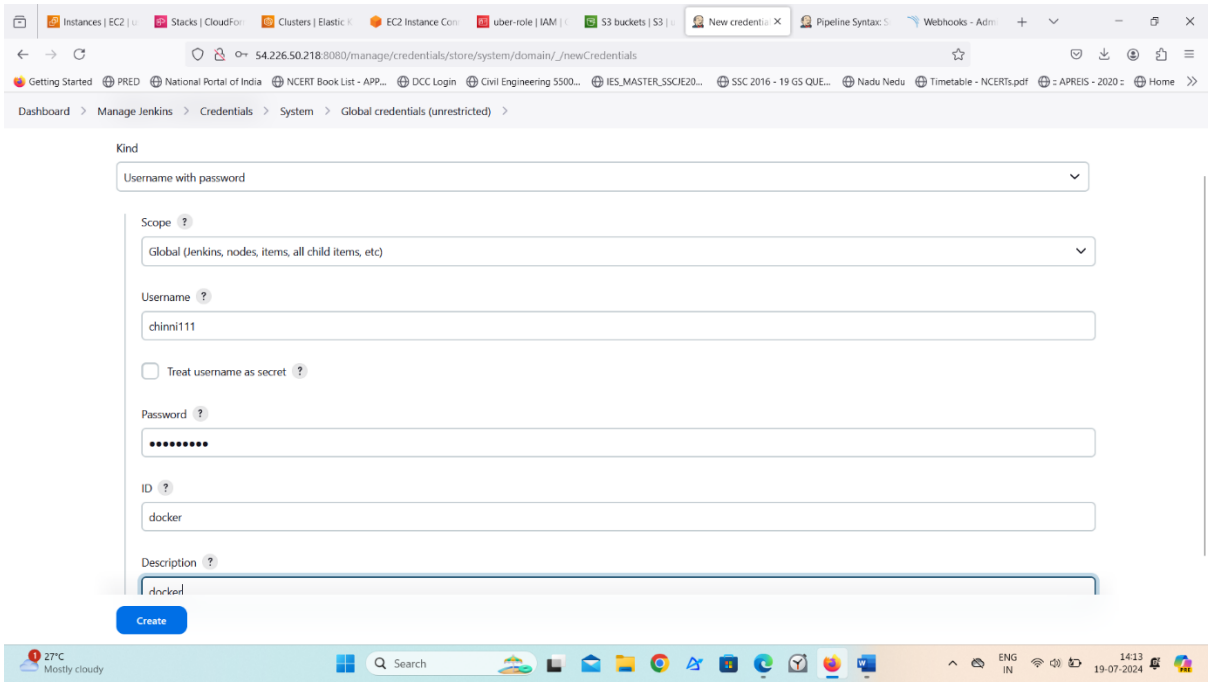


Now add Docker credentials to the Jenkins to log in and push the image

Manage Jenkins → Credentials → global → add credential



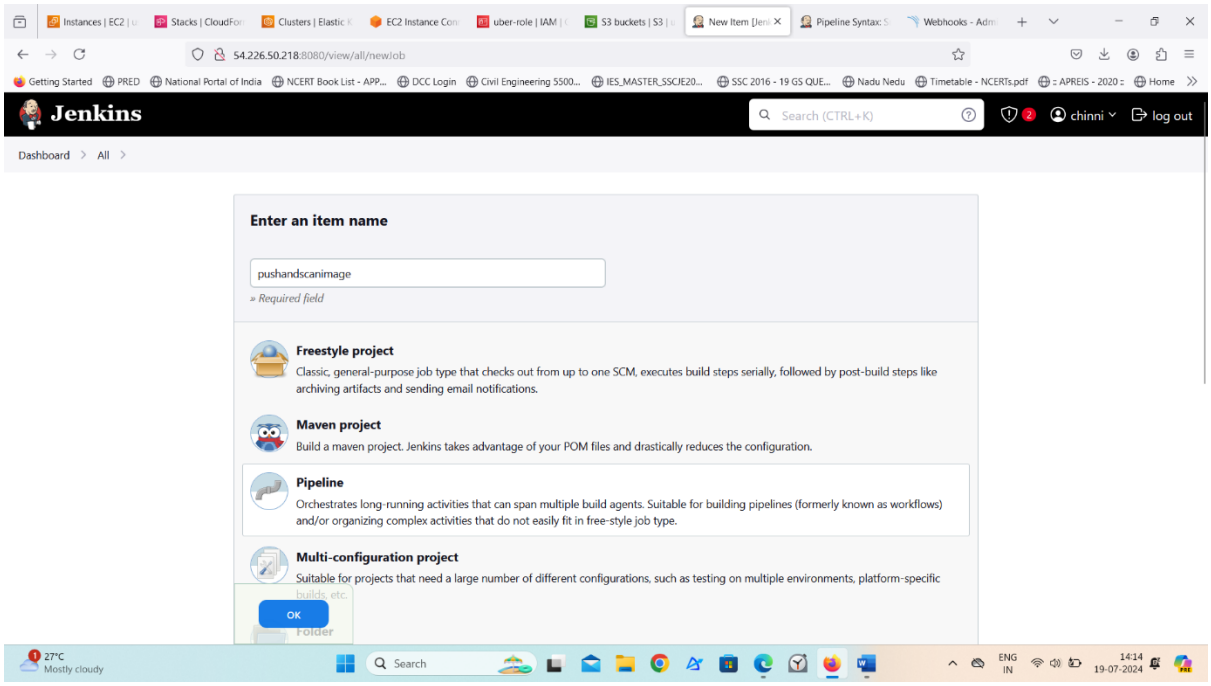
Add Docker Hub Username and Password under Global Credentials



Click on Create.

Step10: Pipeline up to Docker

Now let’s create a new job for our pipeline



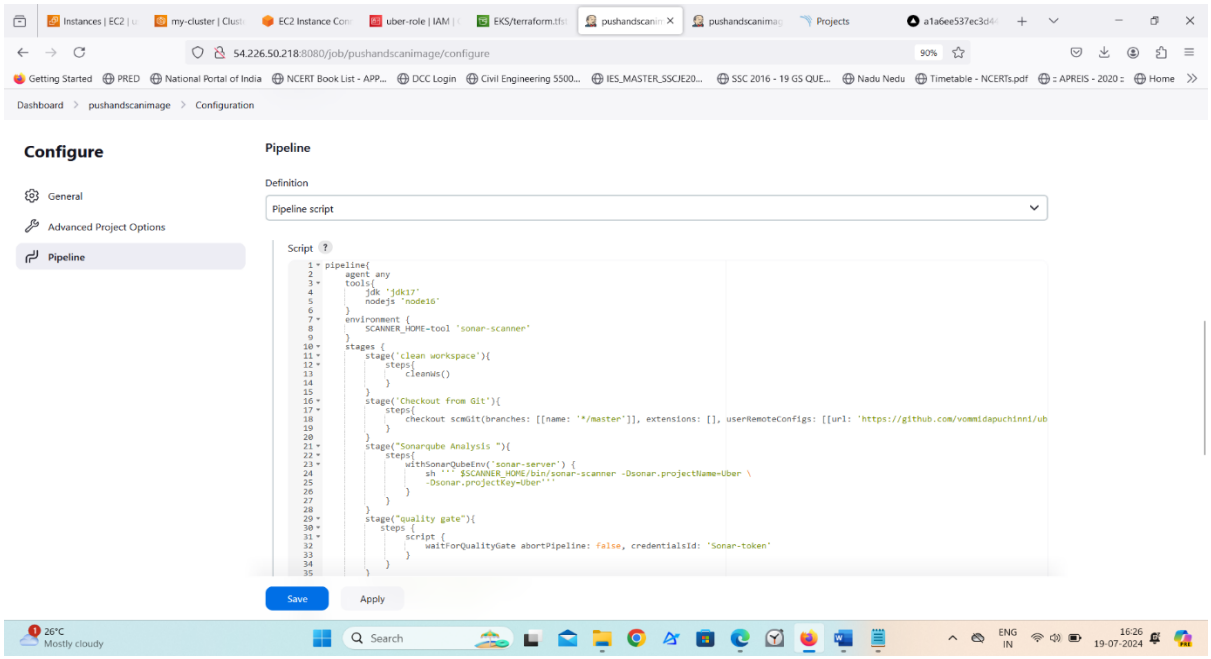
Add this to Pipeline

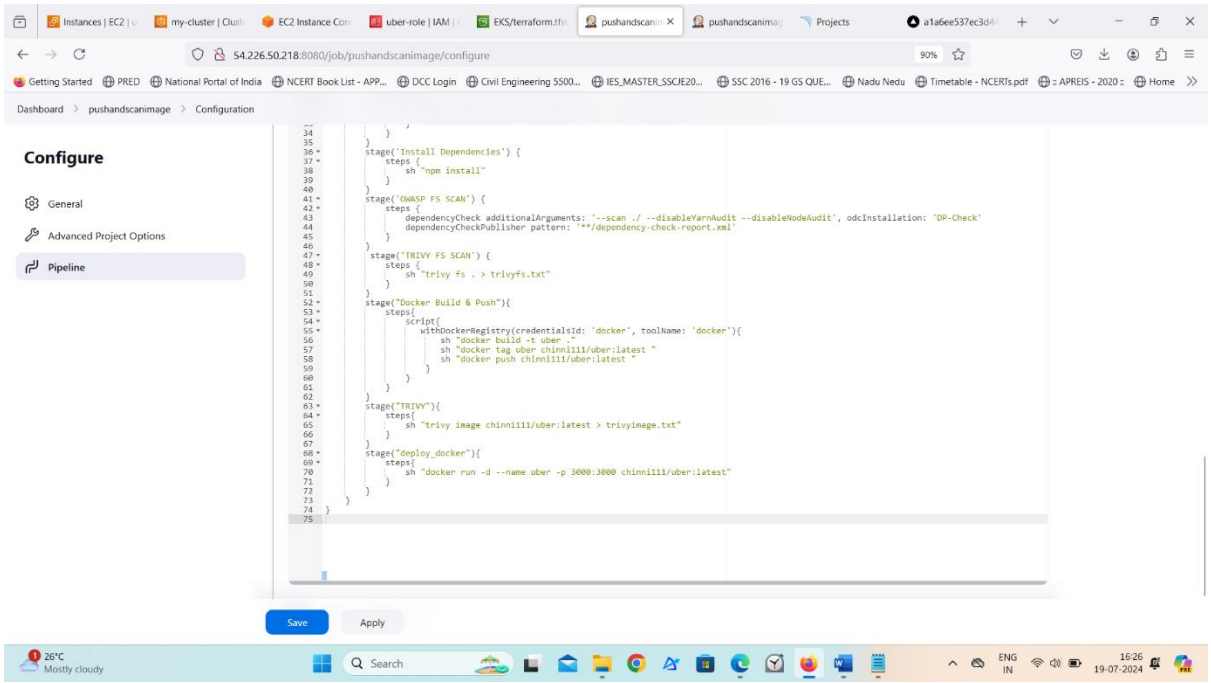
```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout from Git'){
```

```
        steps{
            checkout scmGit(branches: [[name: '*/master']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/vommidapuchinni/uber-clone.git']])
        }
    }
    stage("Sonarqube Analysis "){
        steps{
            withSonarQubeEnv('sonar-server') {
                sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Uber \
                -Dsonar.projectKey=Uber'''
            }
        }
    }
    stage("quality gate"){
       steps {
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage('Install Dependencies') {
        steps {
            sh "npm install"
        }
    }
    stage('OWASP FS SCAN') {
```

```
        steps {
            dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
      }
       stage('TRIVY FS SCAN') {
        steps {
            sh "trivy fs . > trivyfs.txt"
        }
      }
      stage("Docker Build & Push"){
        steps{
            script{
              withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                  sh "docker build -t uber ."
                  sh "docker tag uber chinni111/uber:latest "
                  sh "docker push chinni111/uber:latest "
               }
            }
        }
      }
      stage("TRIVY"){
        steps{
            sh "trivy image chinni111/uber:latest > trivyimage.txt"
        }
      }
```

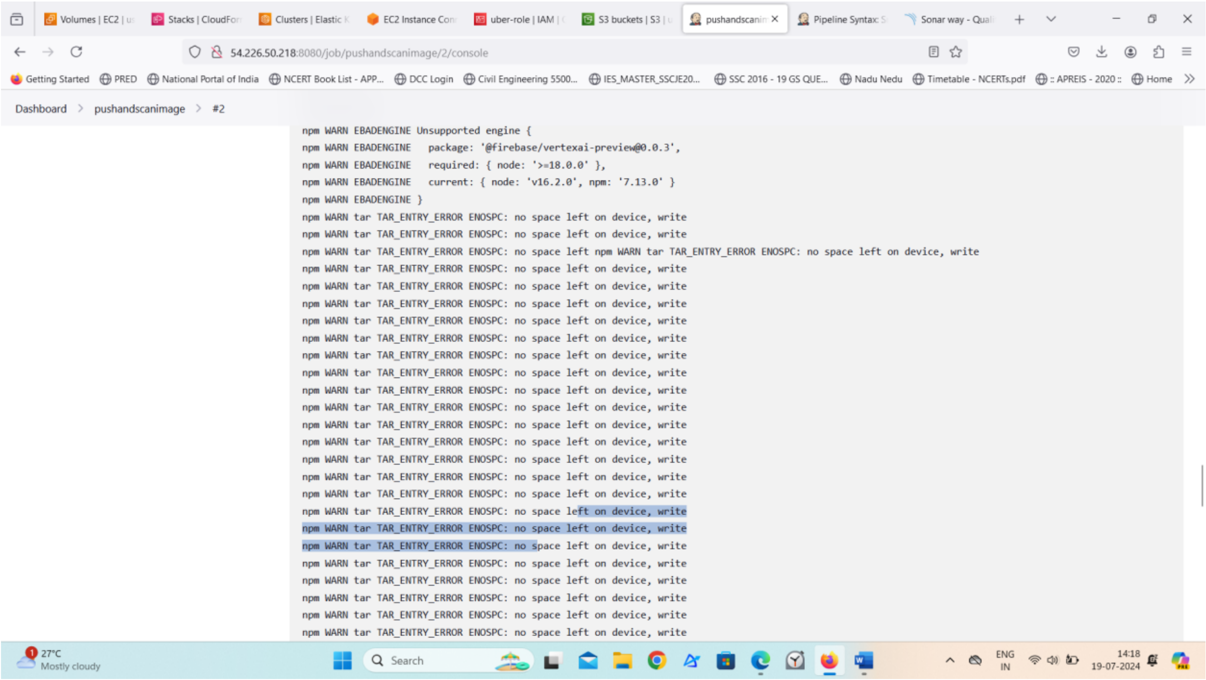
```
        stage("deploy_docker"){
            steps{
                sh "docker run -d --name uber -p 3000:3000 chinni111/uber:latest"
            }
        }
    }
}
```



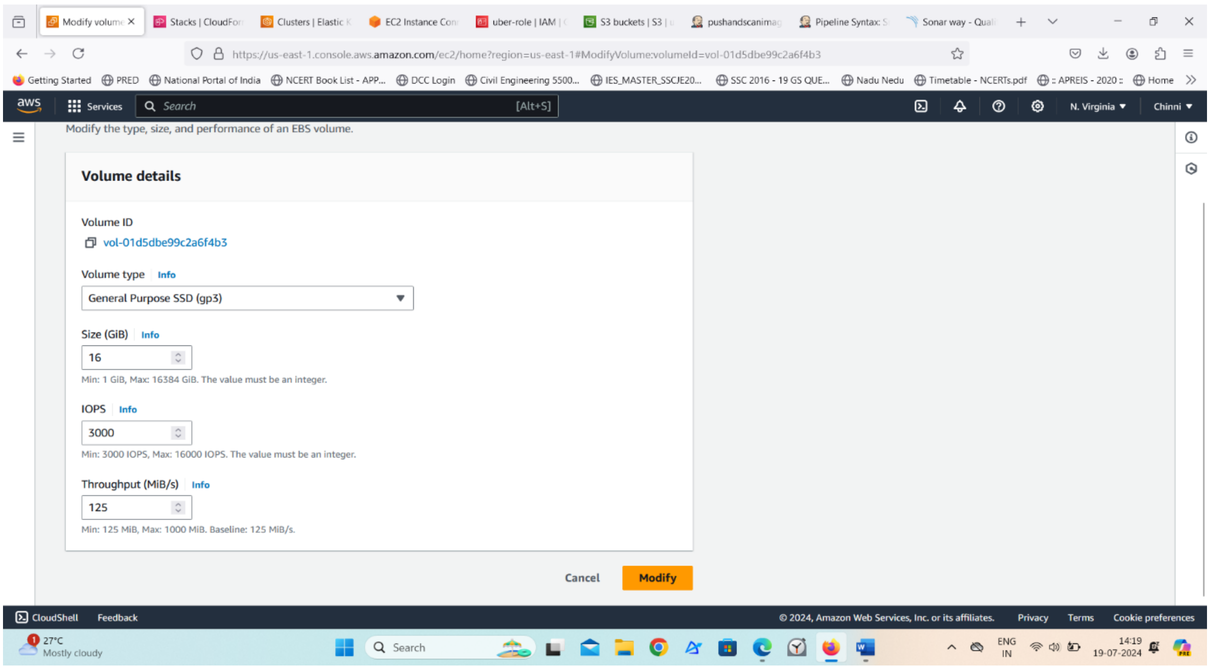


Click on apply and then save it.

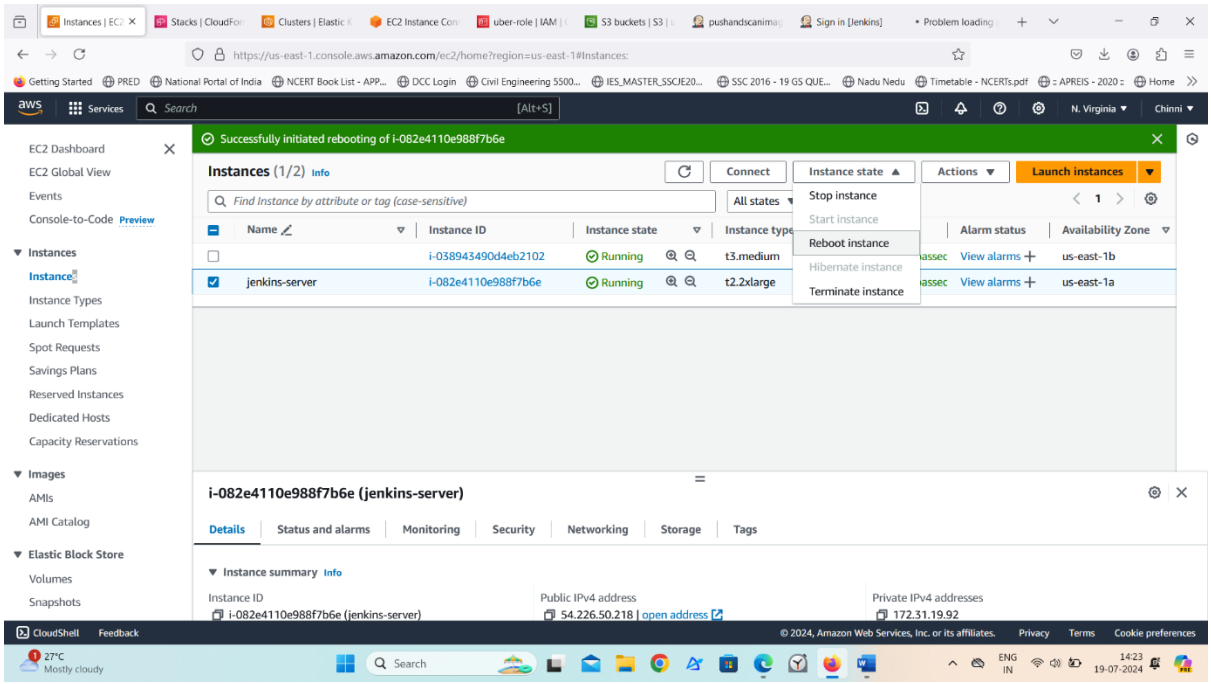
Click on build



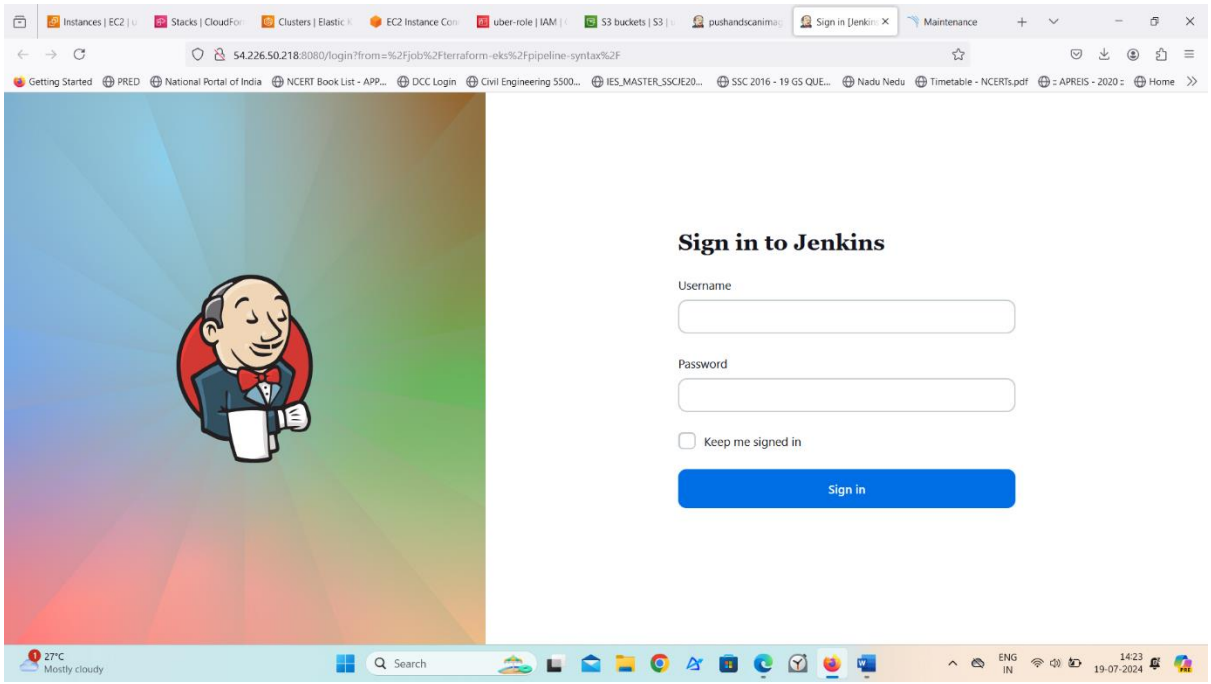
- It showing no space left on my device.
- To increase the storage capacity. Go to volume attached to our ec2 instance.
- Click on volume action.
- Choose modify volume and increase the size of the volume.
- Click on modify.



Now go to instance click on instance state, click on reboot instance.

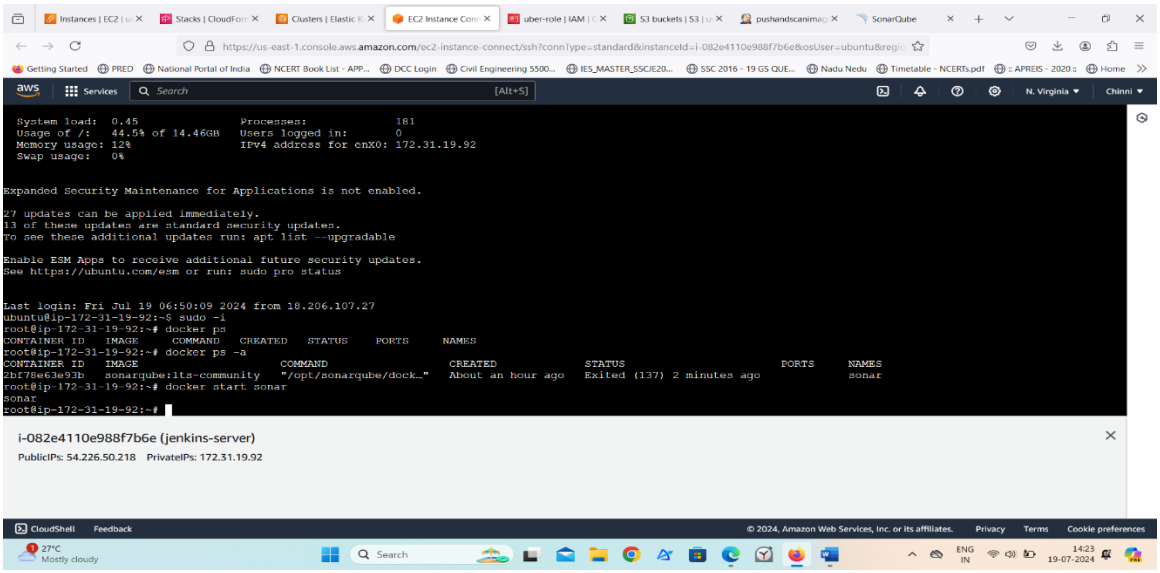


Login to Jenkins.

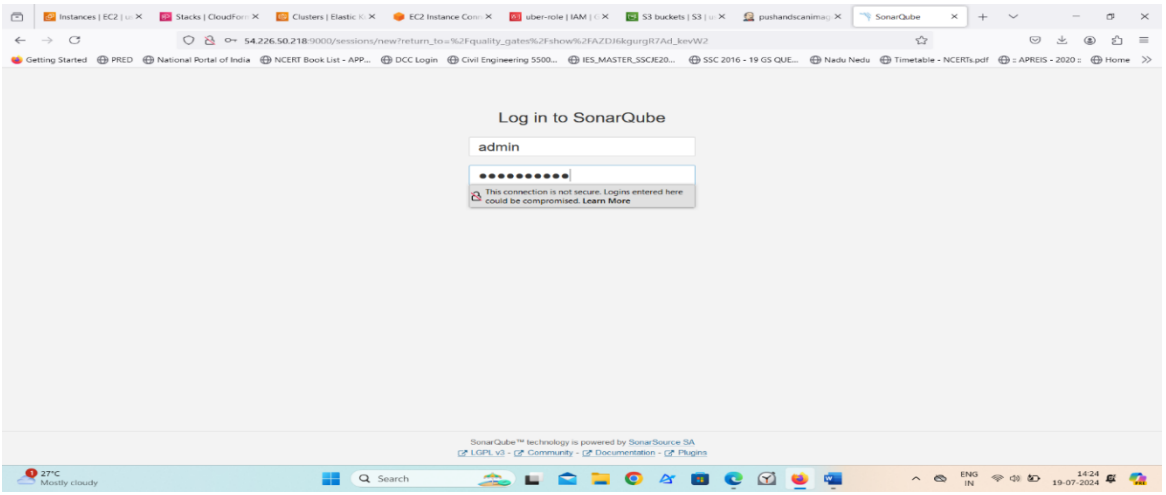


Start sonar server.

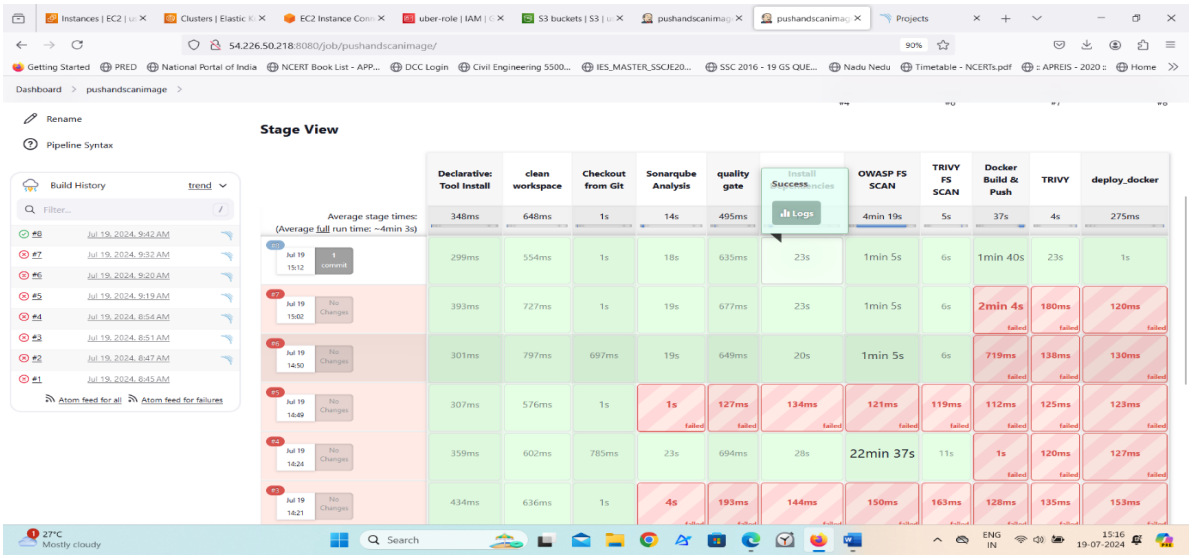
Using docker start sonar

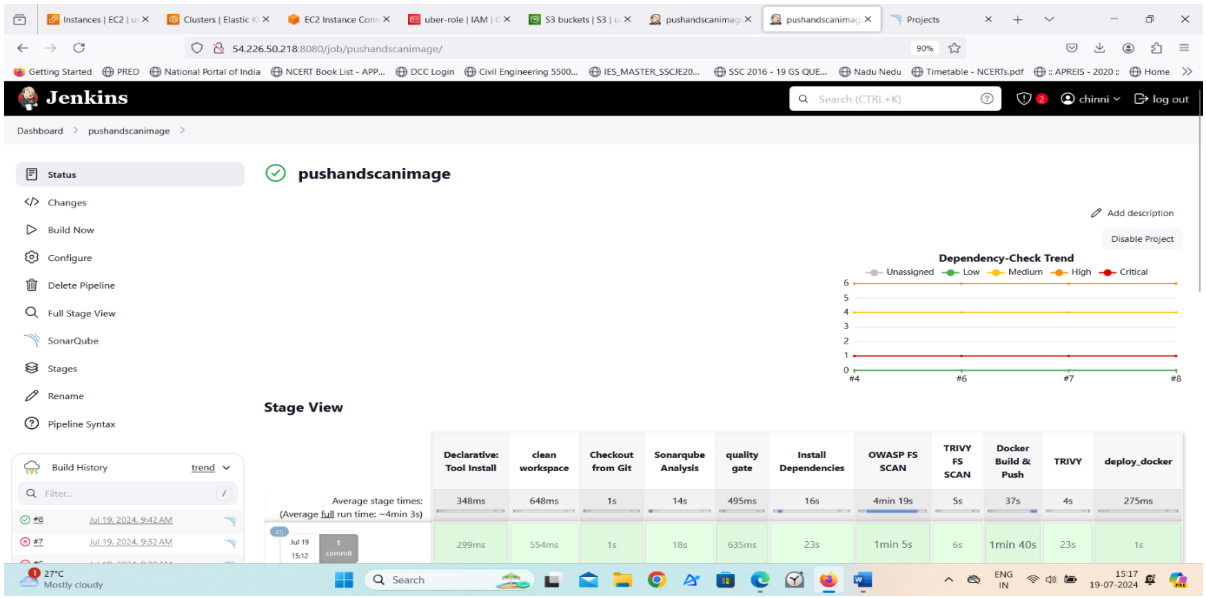


Login to SonarQube server.

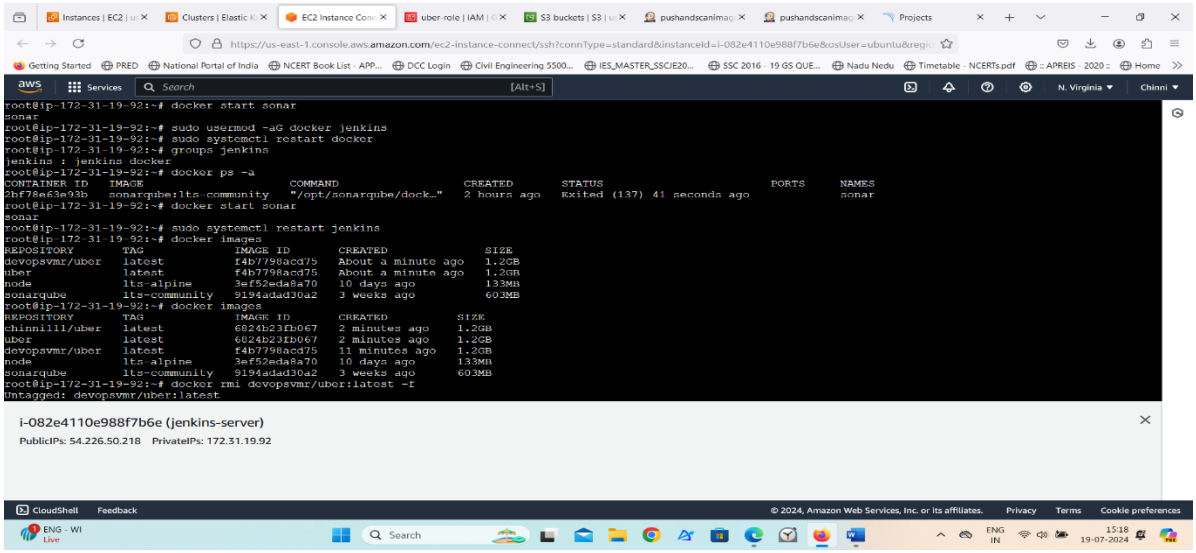


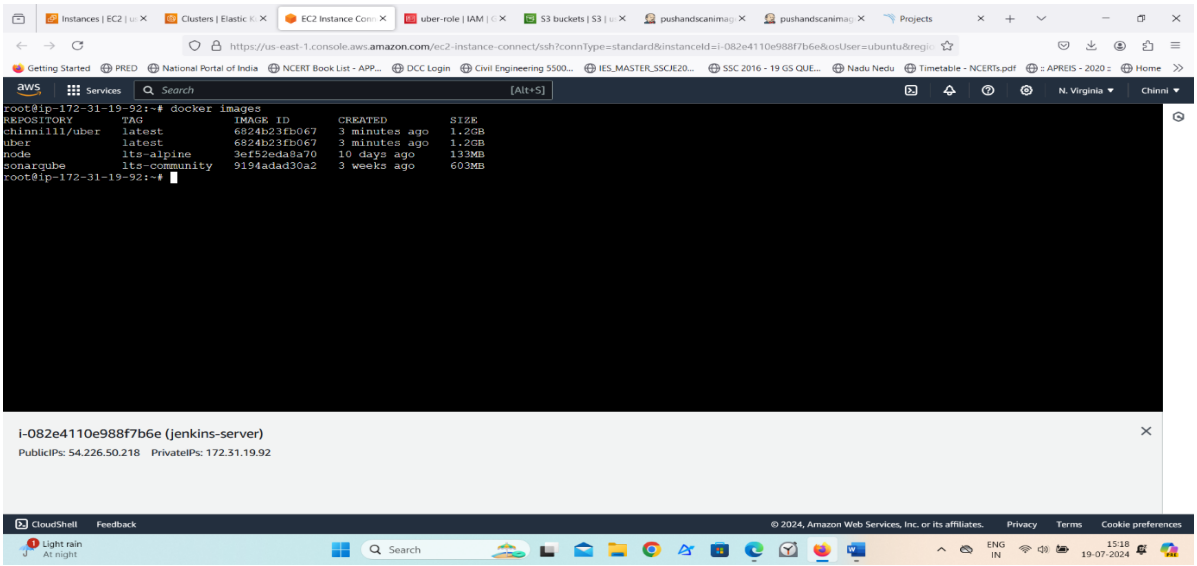
Go to Jenkins and click on our pipeline. Click on build.





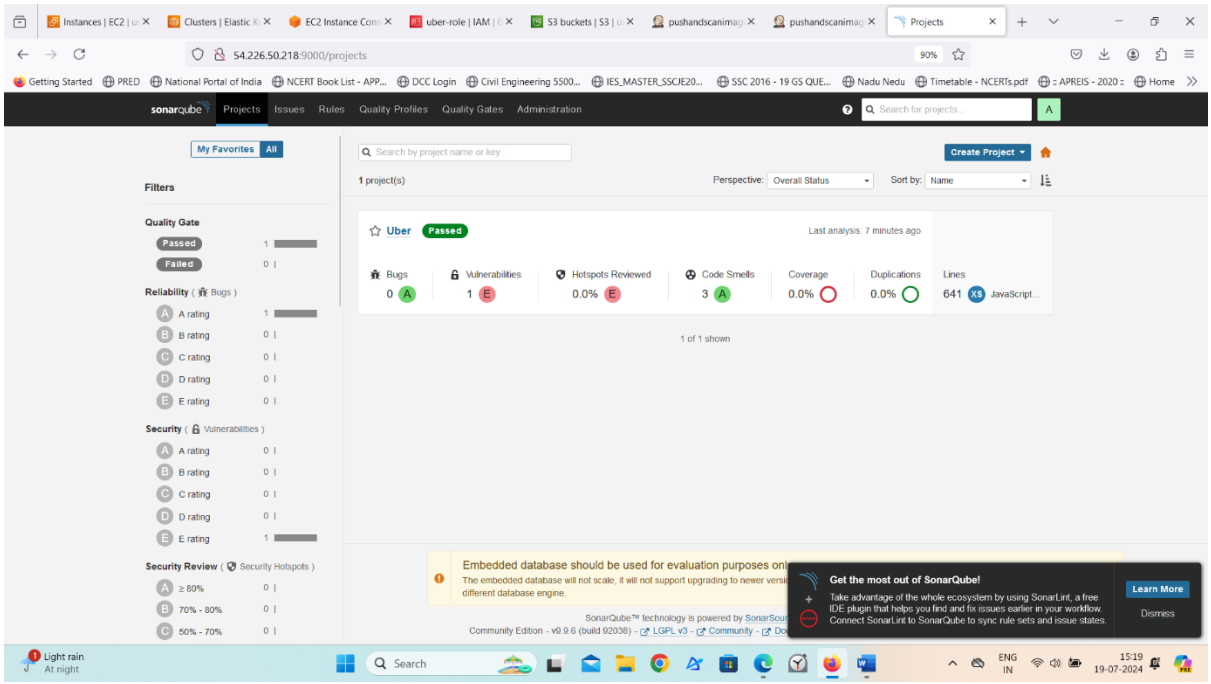
We see our images created.



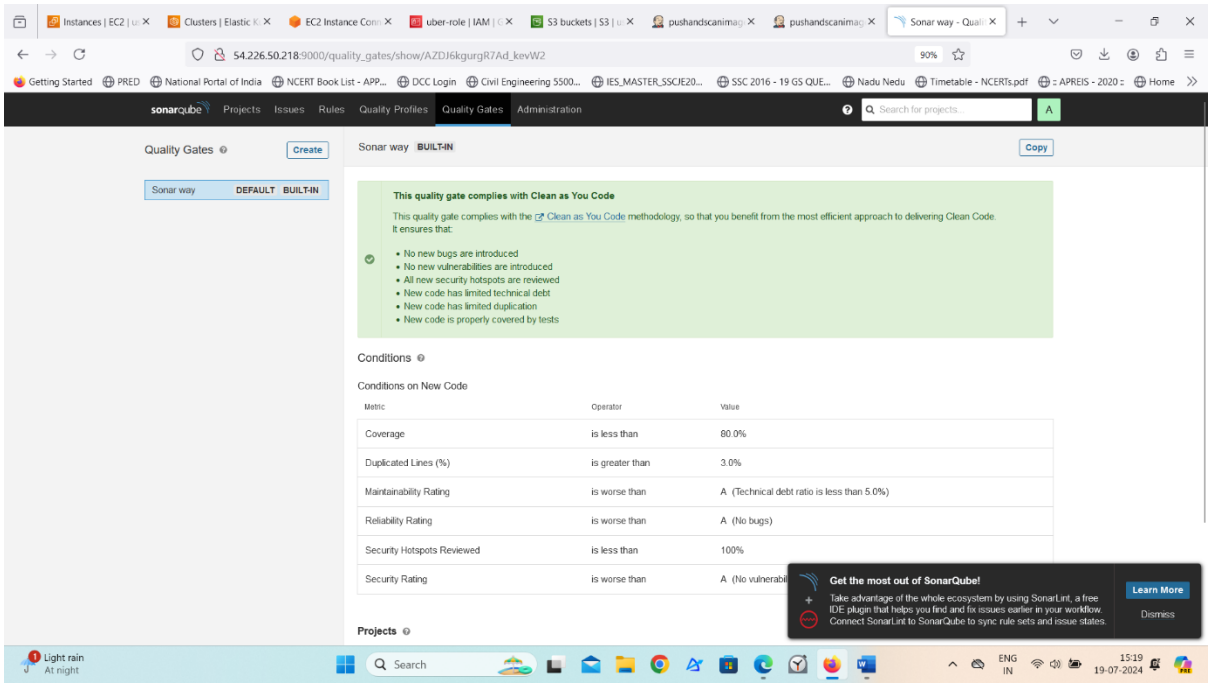


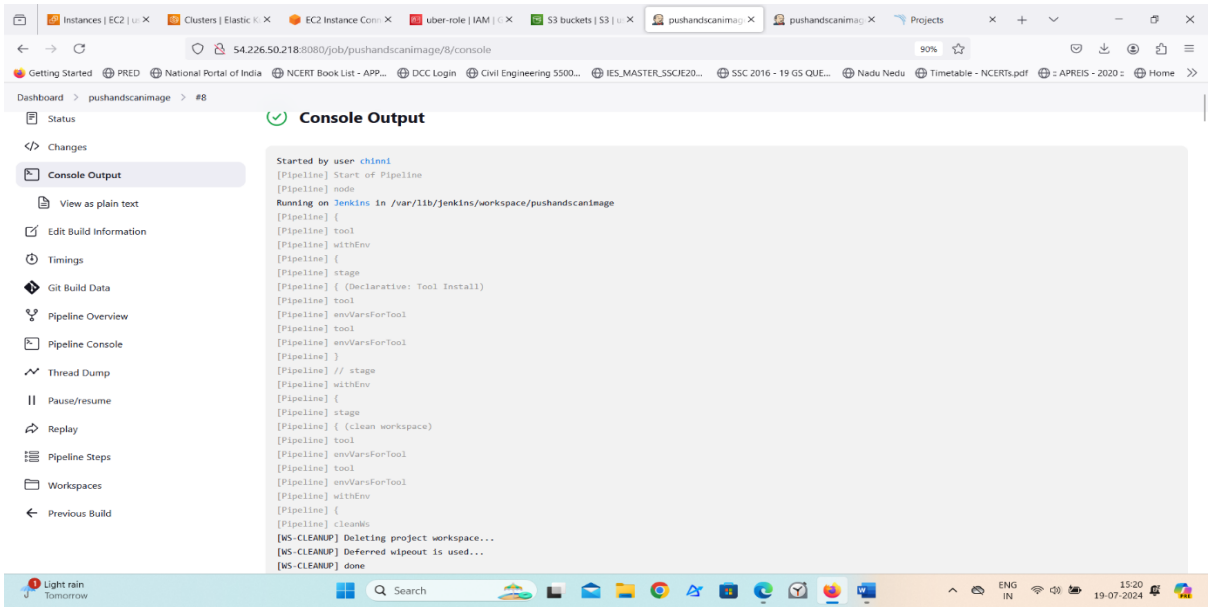
Now go to SonarQube server.

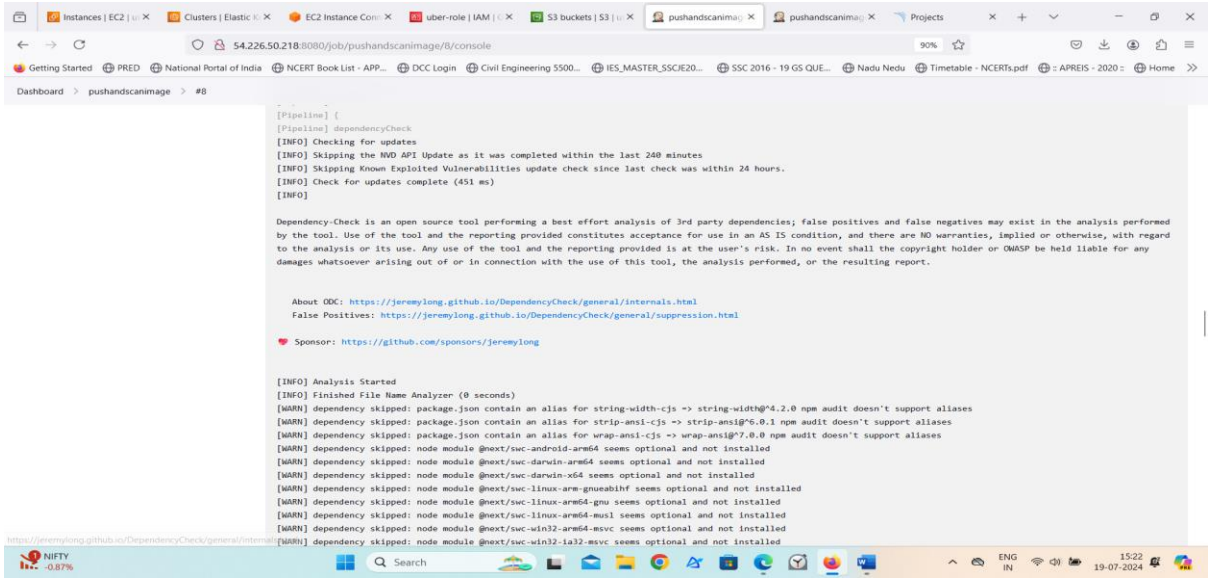
You can see the report has been generated and the status shows as passed. You can see that there are 641 lines it scanned. To see a detailed report, you can go to issues.

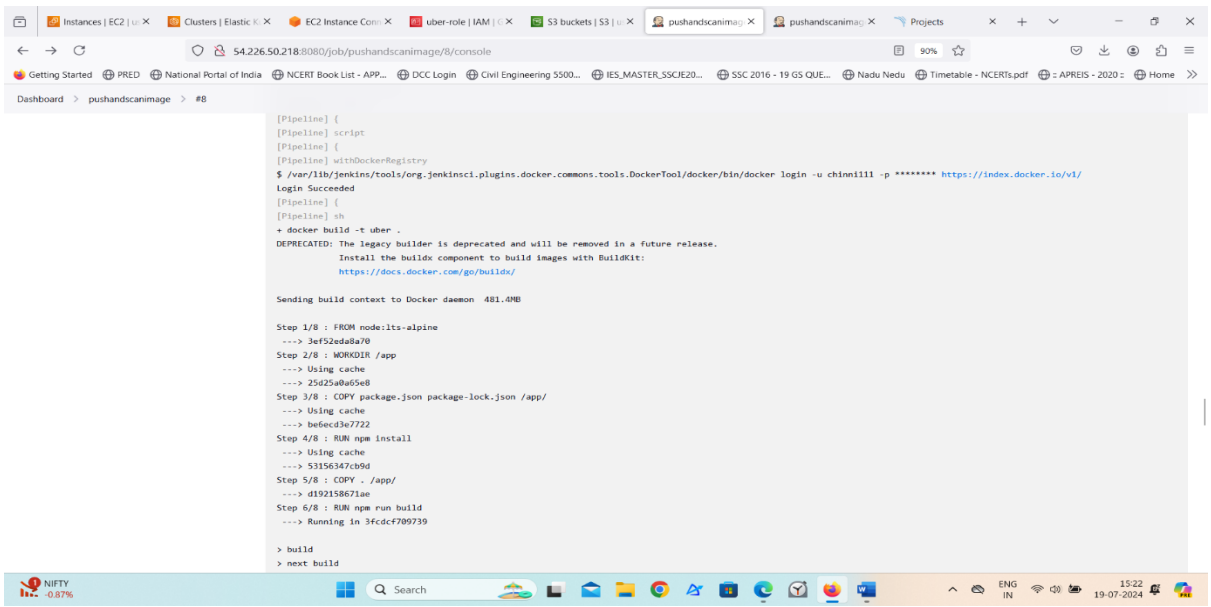


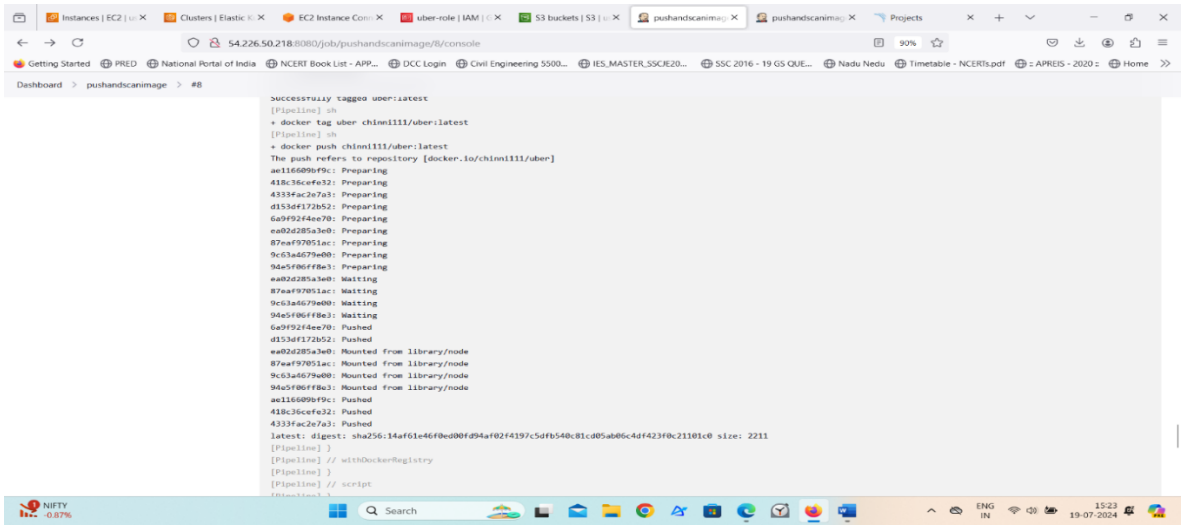
Click on quality gates we see the results and the created quality gate.

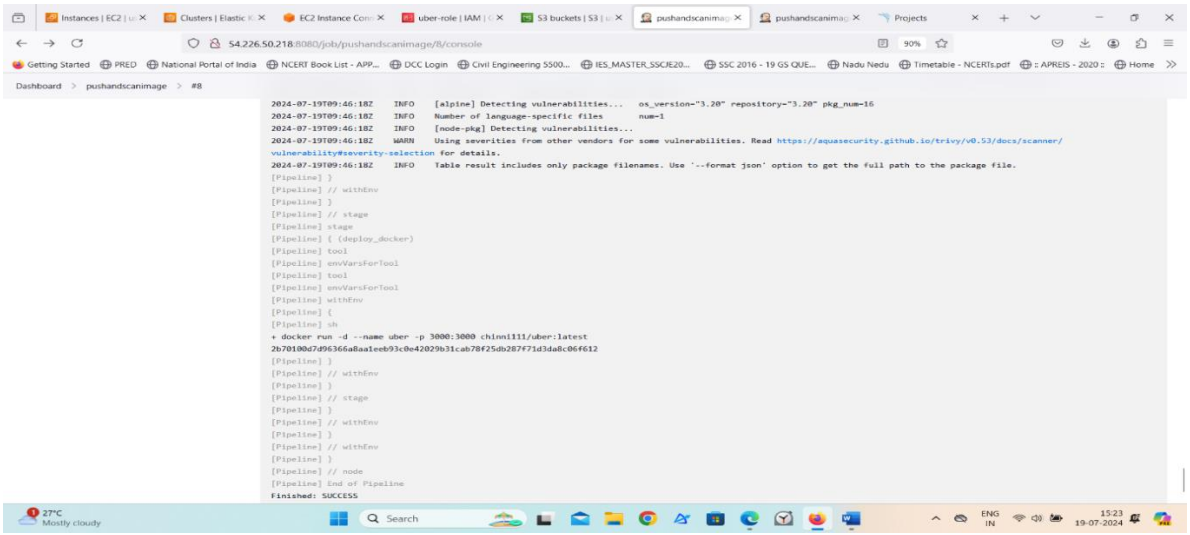



54.226.50.218:8080/job/pushandscanimage/8/console
Dashboard > pushandscanimage > #8
Status
Changes
Console Output
View as plain text
Edit Build Information
Timings
Git Build Data
Pipeline Overview
Pipeline Console
Thread Dump
Pause/resume
Replay
Pipeline Steps
Workspaces
Previous Build
Console Output
Started by user chinni
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pushandscanimage
[Pipeline] {
[Pipeline] tool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Tool Install)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (clean workspace)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] cleanWs
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] done
Light rain
Tomorrow
Search
ENG
IN
15:20
19-07-2024

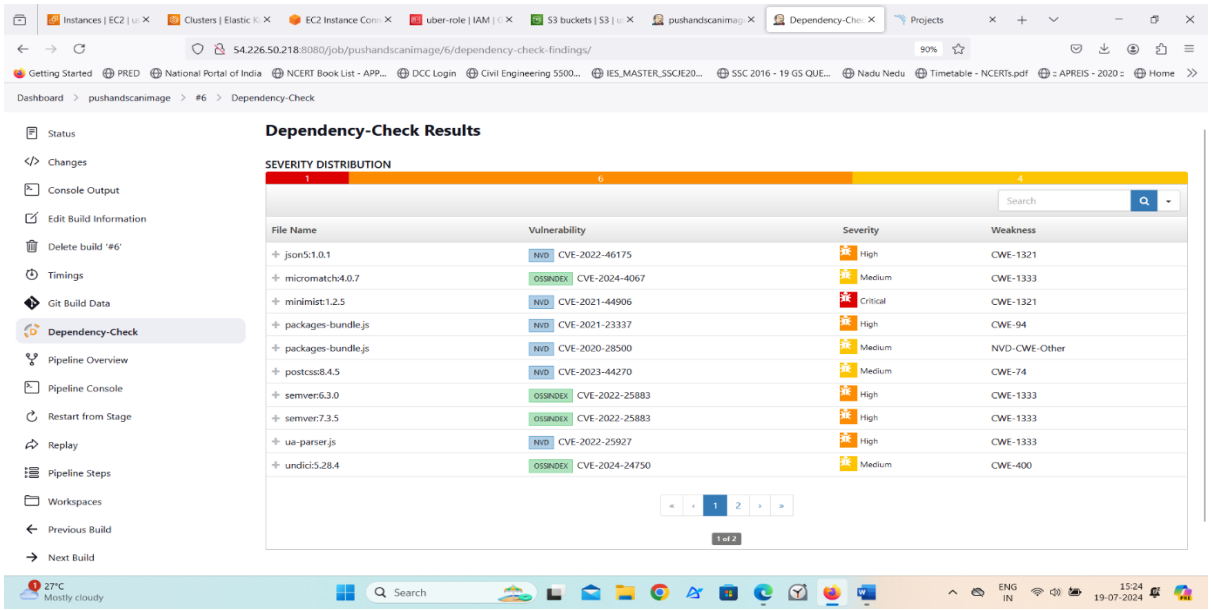

54.226.50.218:8080/job/pushandscanimage/8/console
Dashboard > pushandscanimage > #8
[Pipeline] {
[Pipeline] dependencyCheck
[INFO] Checking for updates
[INFO] Skipping the NVD API Update as it was completed within the last 240 minutes
[INFO] Skipping Known Exploited Vulnerabilities update check since last check was within 24 hours.
[INFO] Check for updates complete (451 ms)
[INFO]
Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.
About ODC: https://jeremylong.github.io/DependencyCheck/general/internals.html
False Positives: https://jeremylong.github.io/DependencyCheck/general/suppression.html
Sponsor: https://github.com/sponsors/jeremylong
[INFO] Analysis Started
[INFO] Finished File Name Analyzer (0 seconds)
[WARN] dependency skipped: package.json contain an alias for string-width-cjs => string-width@^4.2.0 npm audit doesn't support aliases
[WARN] dependency skipped: package.json contain an alias for strip-ansi-cjs => strip-ansi@^6.0.1 npm audit doesn't support aliases
[WARN] dependency skipped: package.json contain an alias for wrap-ansi-cjs => wrap-ansi@^7.0.0 npm audit doesn't support aliases
[WARN] dependency skipped: node module @next/swc-android-arm64 seems optional and not installed
[WARN] dependency skipped: node module @next/swc-darwin-arm64 seems optional and not installed
[WARN] dependency skipped: node module @next/swc-darwin-x64 seems optional and not installed
[WARN] dependency skipped: node module @next/swc-linux-arm-gnueabihf seems optional and not installed
[WARN] dependency skipped: node module @next/swc-linux-arm64-gnu seems optional and not installed
[WARN] dependency skipped: node module @next/swc-linux-arm64-musl seems optional and not installed
[WARN] dependency skipped: node module @next/swc-win32-arm64-msvc seems optional and not installed
[WARN] dependency skipped: node module @next/swc-win32-ia32-msvc seems optional and not installed
NIFTY
-0.87%
Search
ENG
IN
15:22
19-07-2024

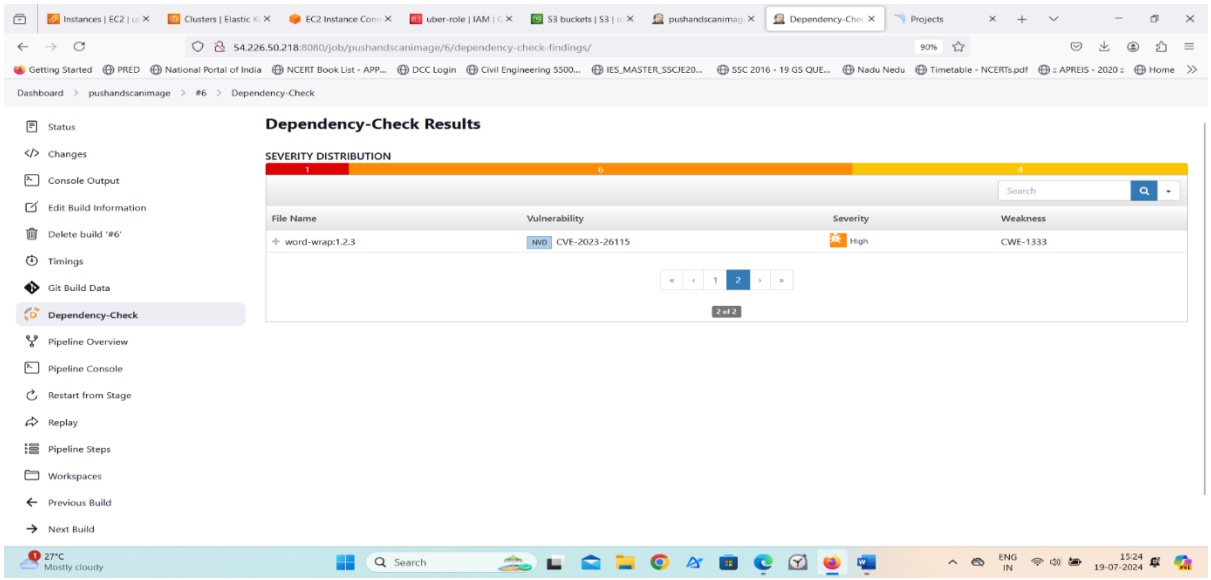

54.226.50.218:8080/job/pushandscanimage/8/console
Dashboard > pushandscanimage > #8
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] withDockerRegistry
$ /var/lib/jenkins/tools/org.jenkinsci.plugins.docker.commons.tools.DockerTool/docker/bin/docker login -u chinni111 -p ******** https://index.docker.io/v1/
Login Succeeded
[Pipeline] {
[Pipeline] sh
+ docker build -t uber .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/
Sending build context to Docker daemon 481.4MB
Step 1/8 : FROM node:lts-alpine
---> 3ef52eda8a70
Step 2/8 : WORKDIR /app
---> Using cache
---> 25d25a0a65e8
Step 3/8 : COPY package.json package-lock.json /app/
---> Using cache
---> be6ecd3e7722
Step 4/8 : RUN npm install
---> Using cache
---> 53156347cb9d
Step 5/8 : COPY . /app/
---> d192158671ae
Step 6/8 : RUN npm run build
---> Running in 3fcdcf709739
> build
> next build
NIFTY
-0.87%
Search
ENG
IN
15:22
19-07-2024



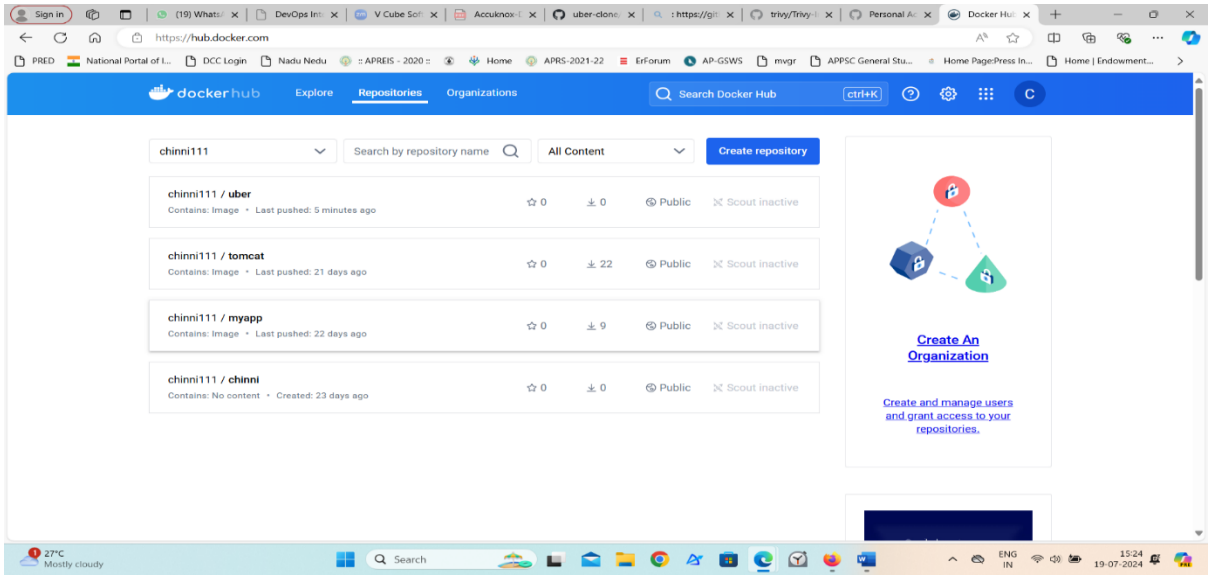


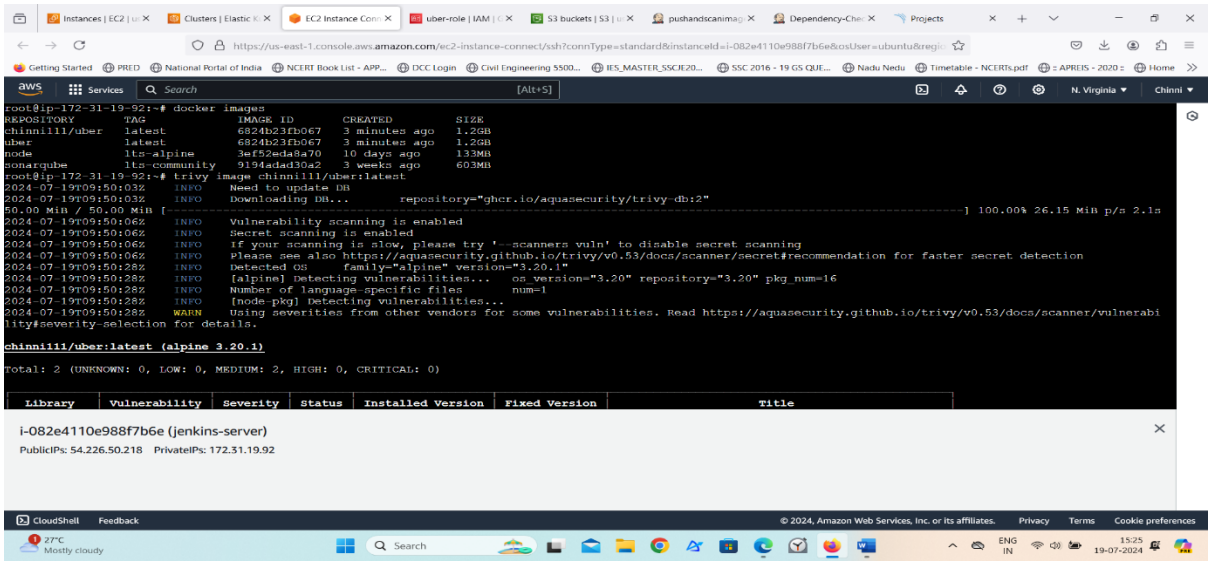
OWASP, you will see that in status, a graph will also be generated and Vulnerabilities.



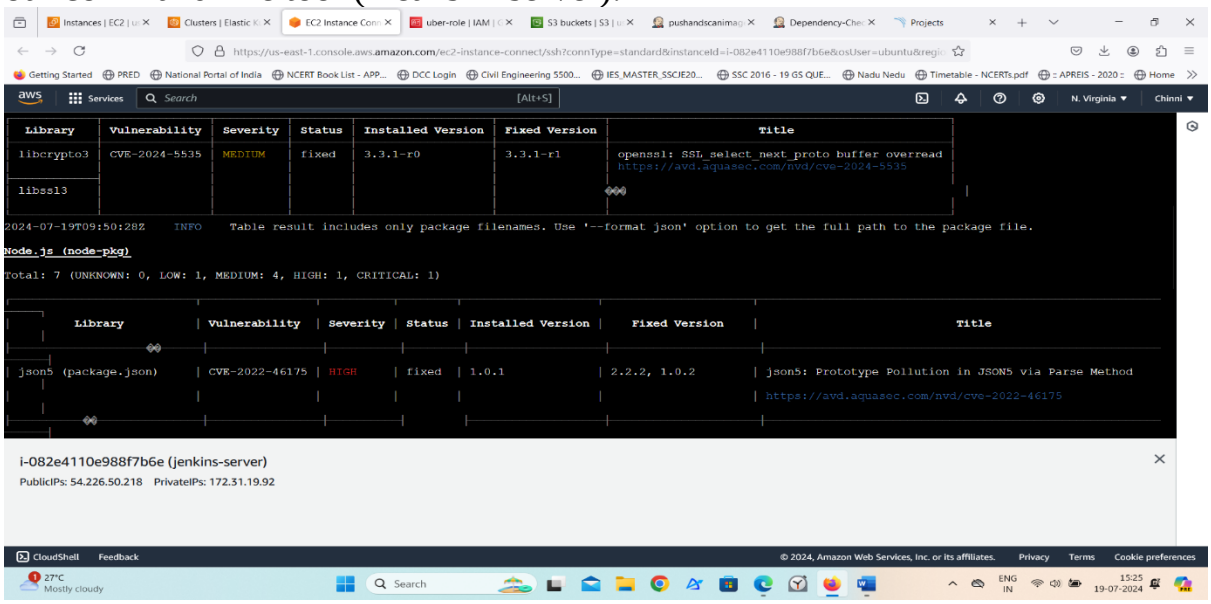


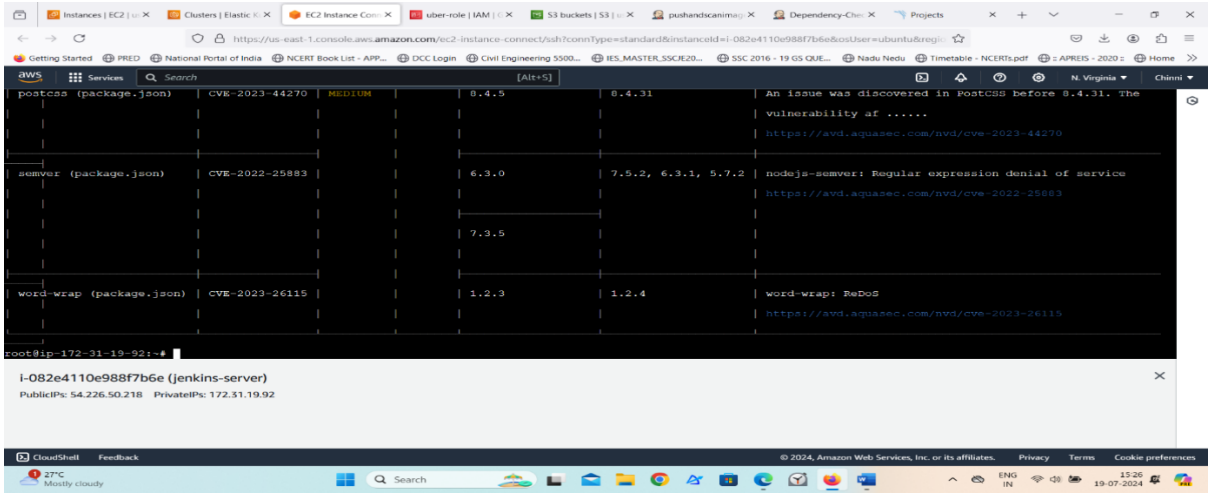
When you log in to Docker hub, you will see a new image is created



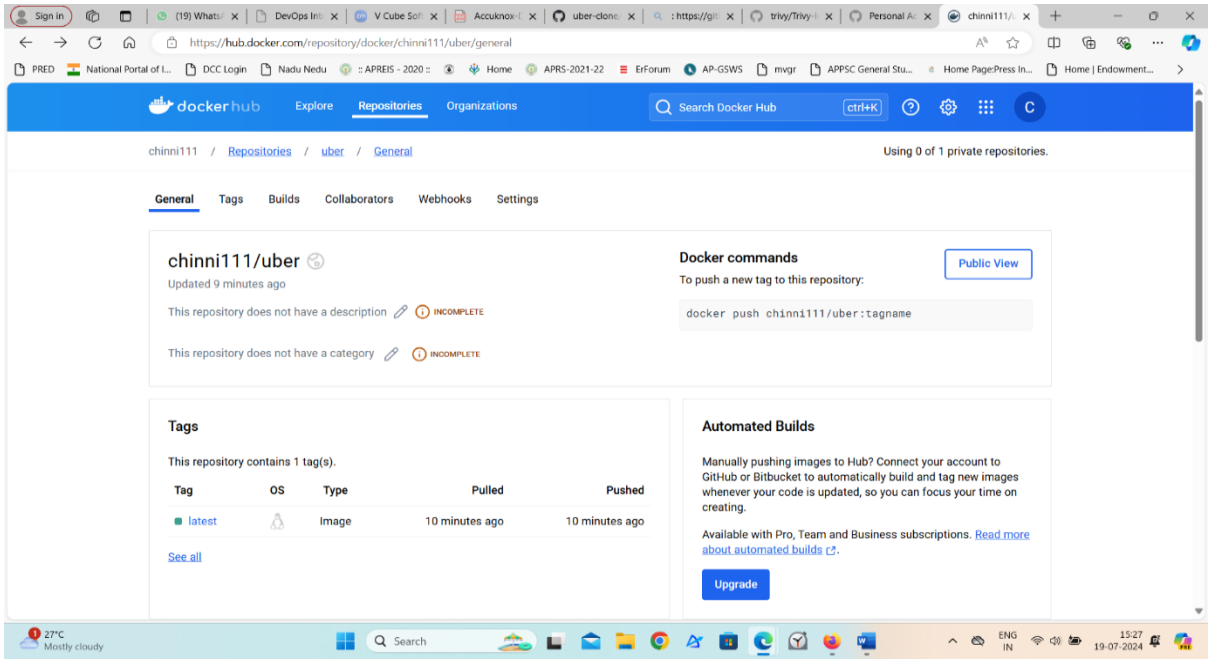


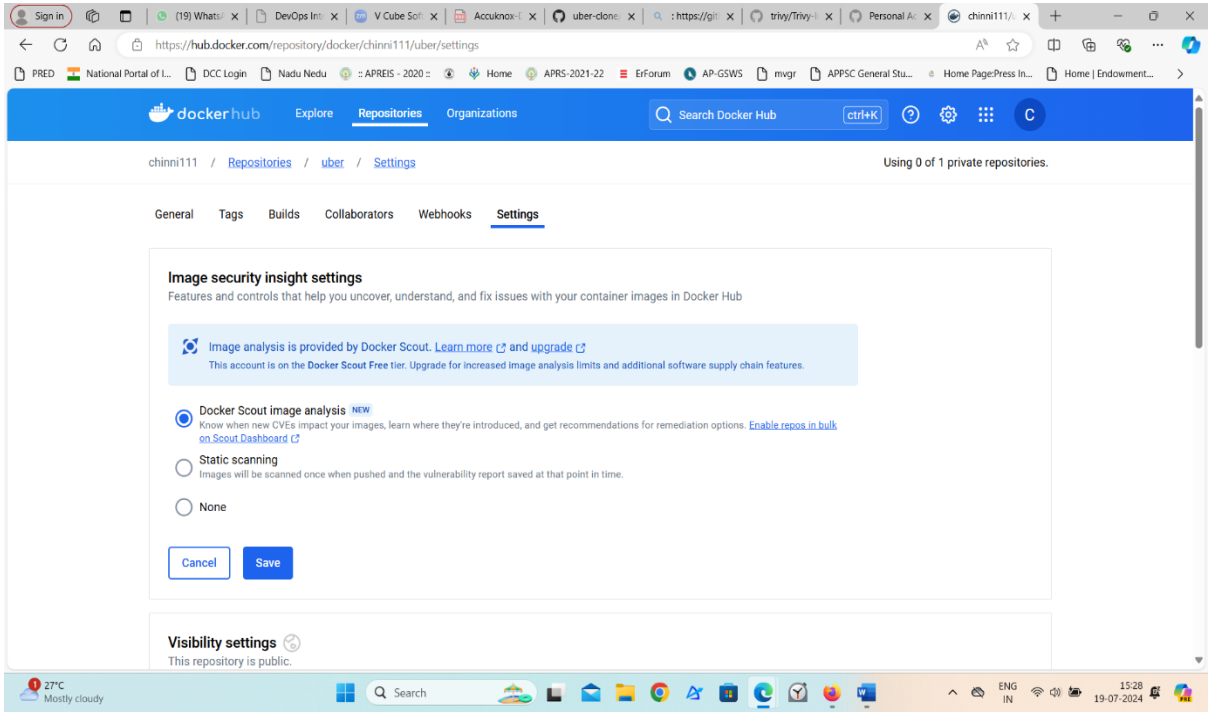
Using trivy image imagename: tag we can see our severity and vulnerability in our command line tool (means in server).





- Using docker scout also we can scan our image.
- Click our pushed repo and go to its settings. Choose docker scout image analysis, click on save

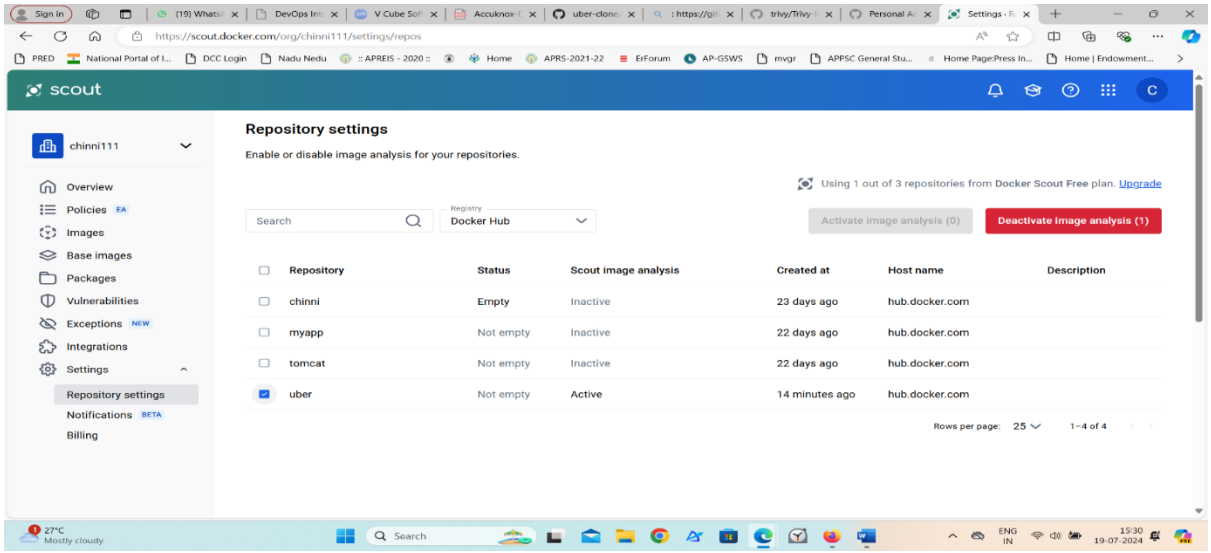




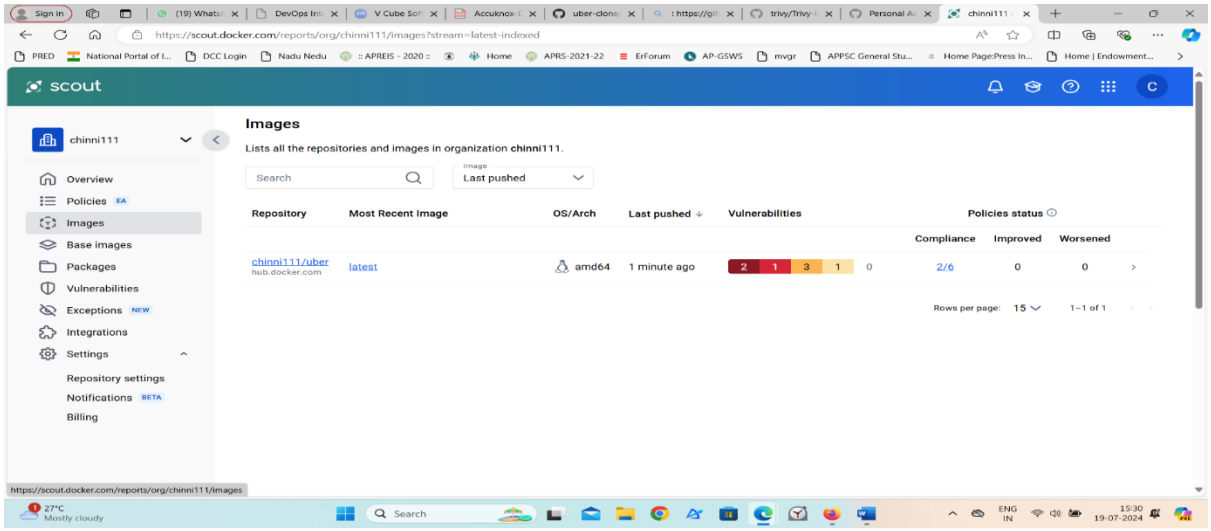
Click on apps symbol on left side of our profile icon.

There we see docker scout click on it skip it for four times.

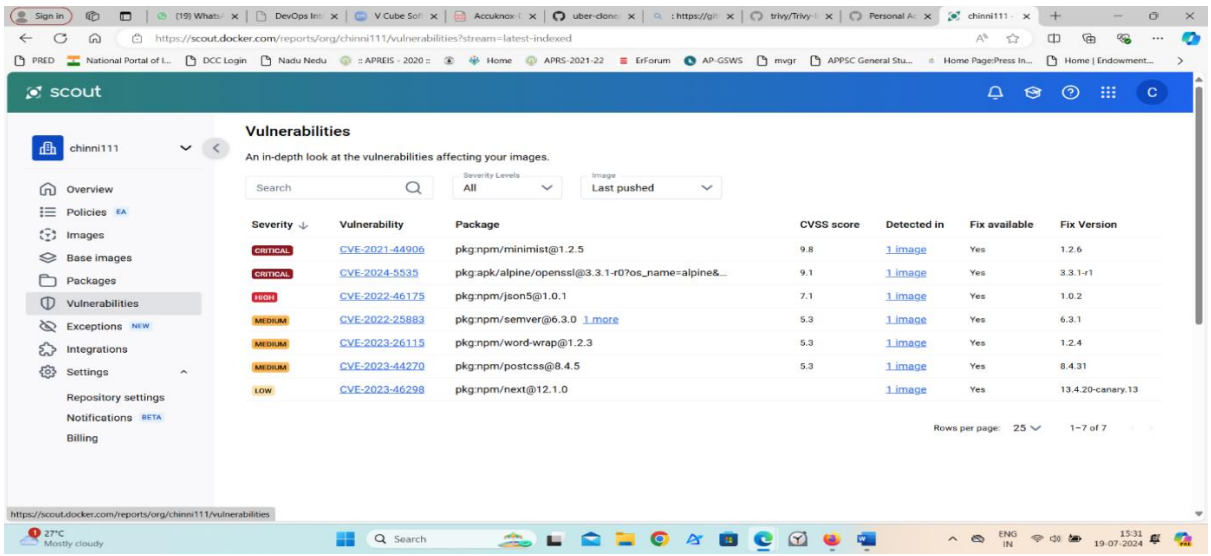
Now go to settings → repo settings choose our image.



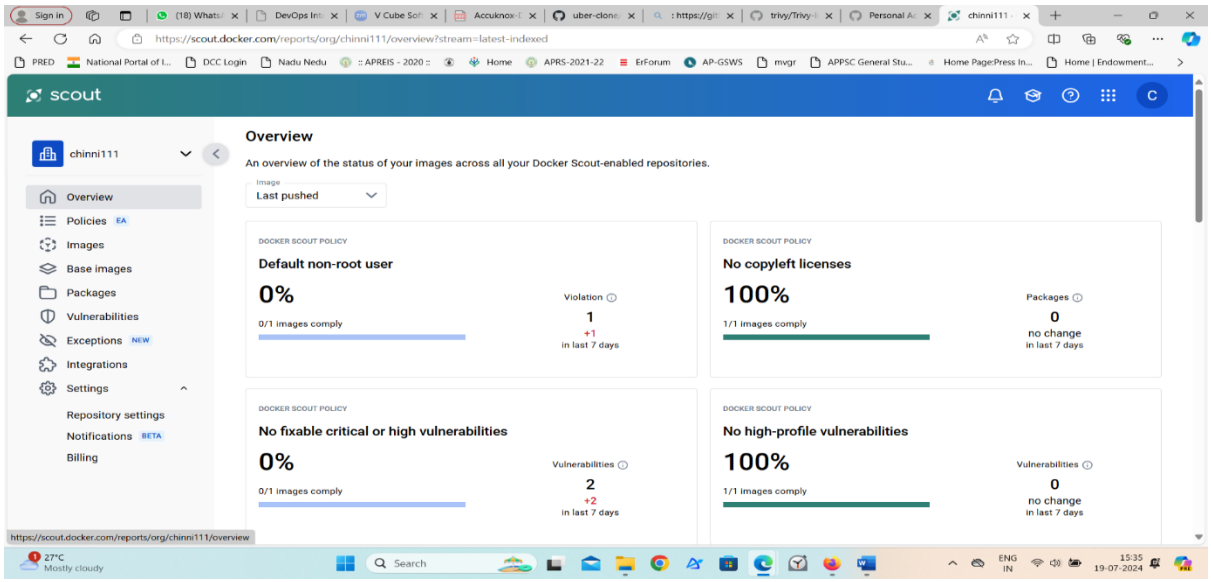
Click on images is left side dashboard. We like below.



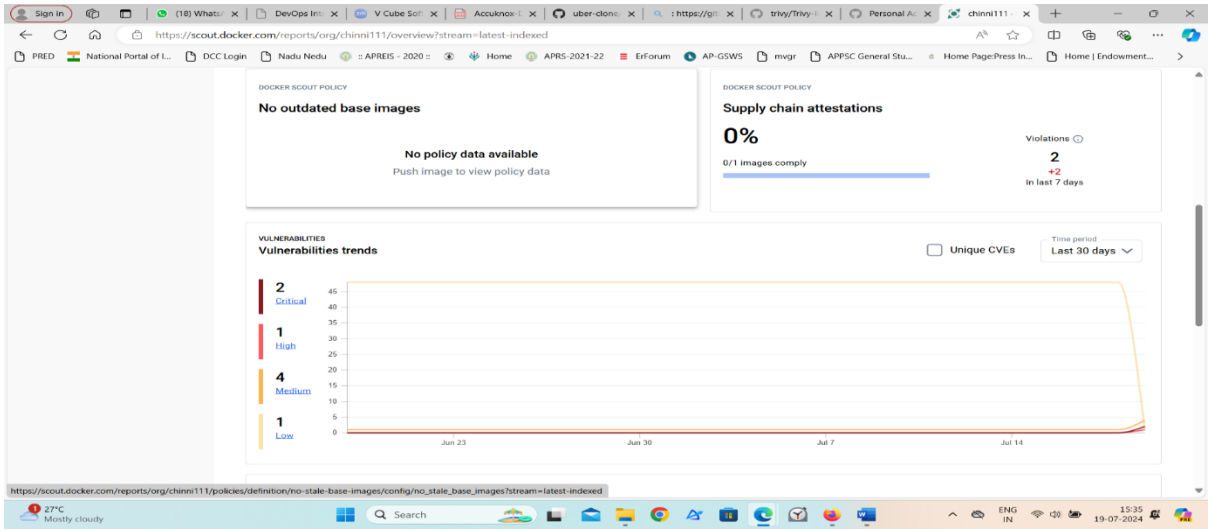
Click on vulnerabilities we see all our image vulnerabilities.

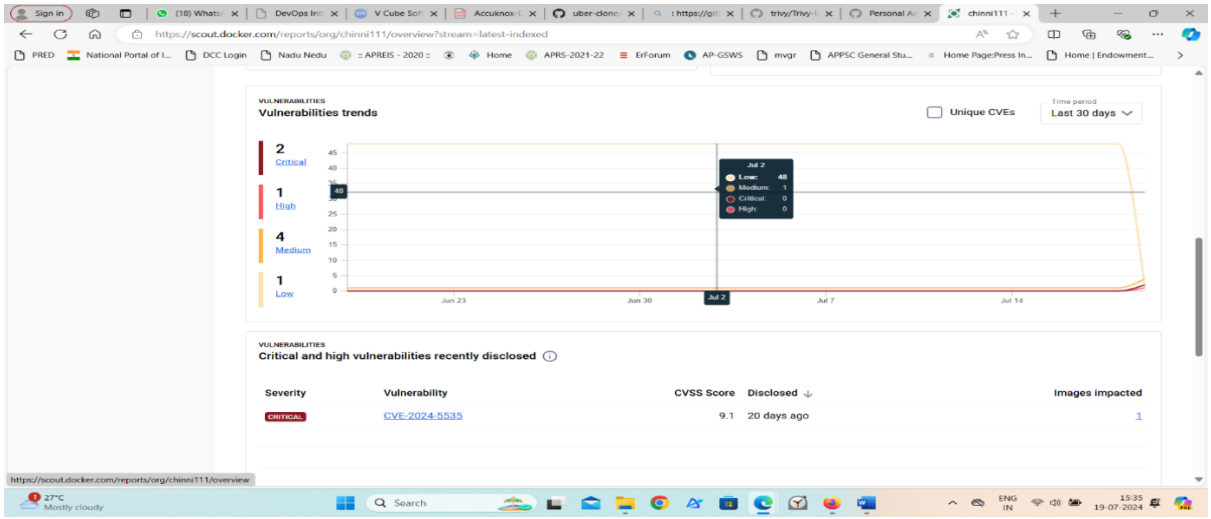


Total overview clicks on overview.



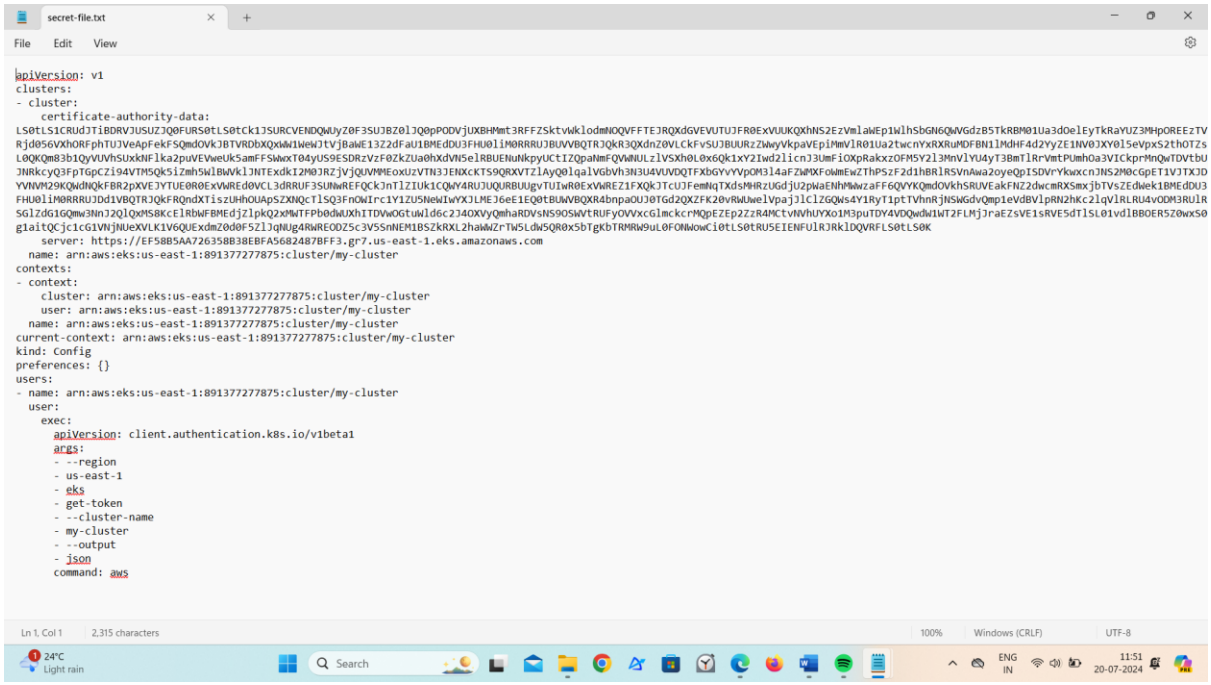
Come down we see vulnerabilities trends

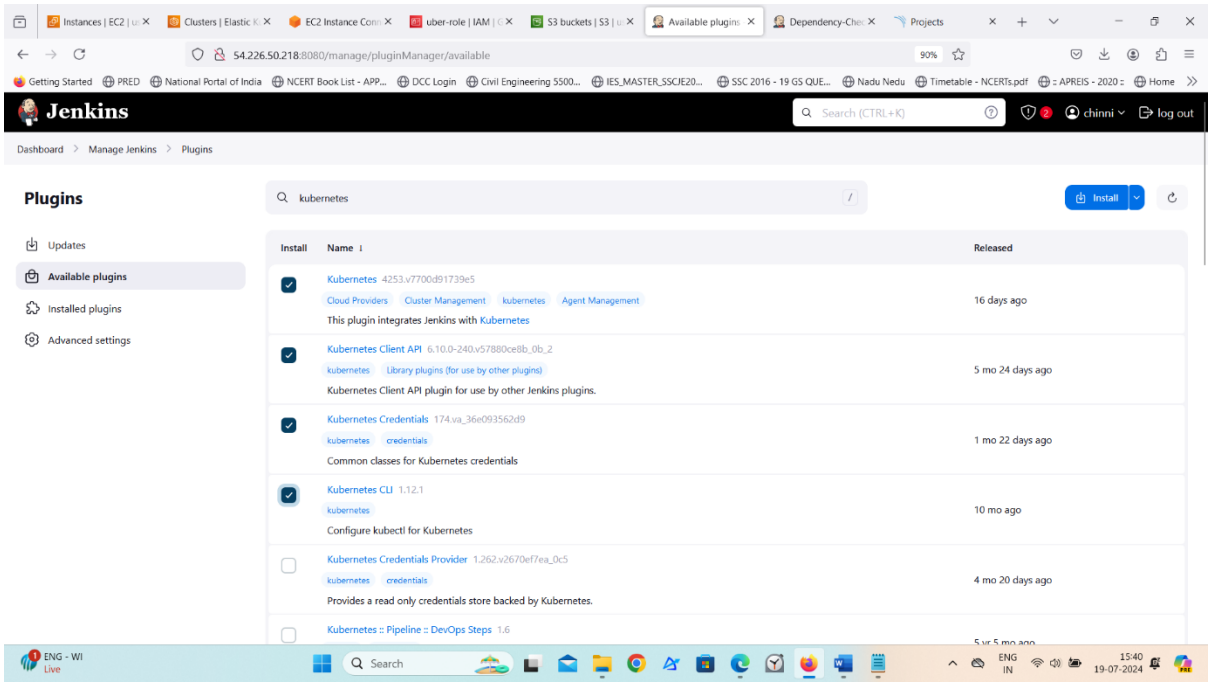




Step11: Kubernetes Deployment

- Go to Putty of your Jenkins instance SSH and enter the below command
- aws eks update-kubeconfig --name <CLUSTER NAME> --region <CLUSTER REGION>
- aws eks update-kubeconfig --name EKS_CLOUD --region ap-south-1
- Let's see the nodes
- kubectl get nodes
- Copy the config file to Jenkins master or the local file manager and save it
- copy it and save it in documents or another folder save it as secret-file.txt
- Note: create a secret-file.txt in your file explorer save the config in it and use this at the kubernetes credential section.
- Install Kubernetes Plugin, once it's installed successfully

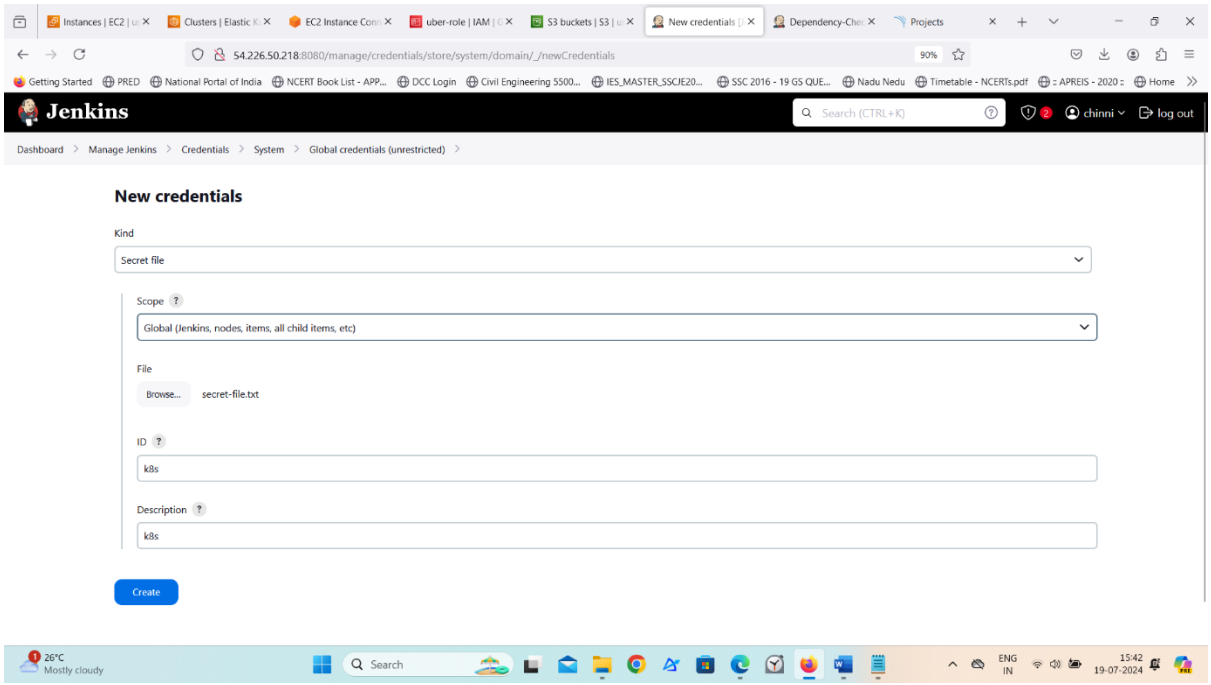


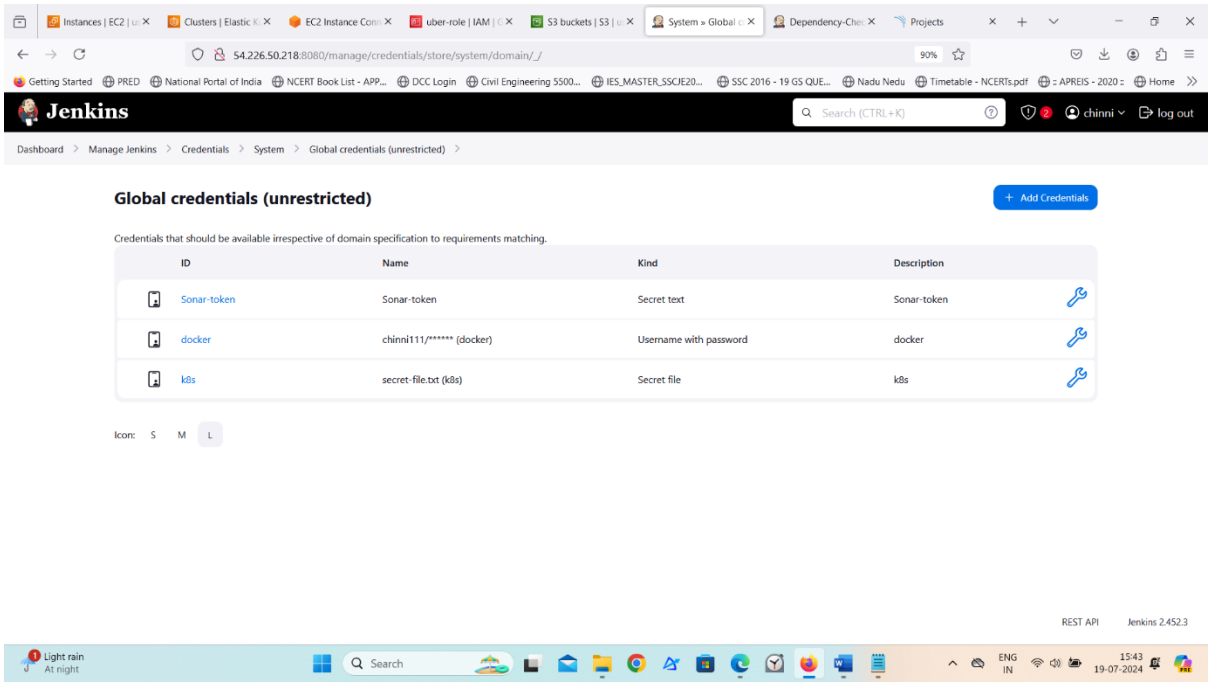


go to manage Jenkins → manage credentials → Click on Jenkins global → add credentials.

Choose kind as secret file. browse the saved file.

Click on create.



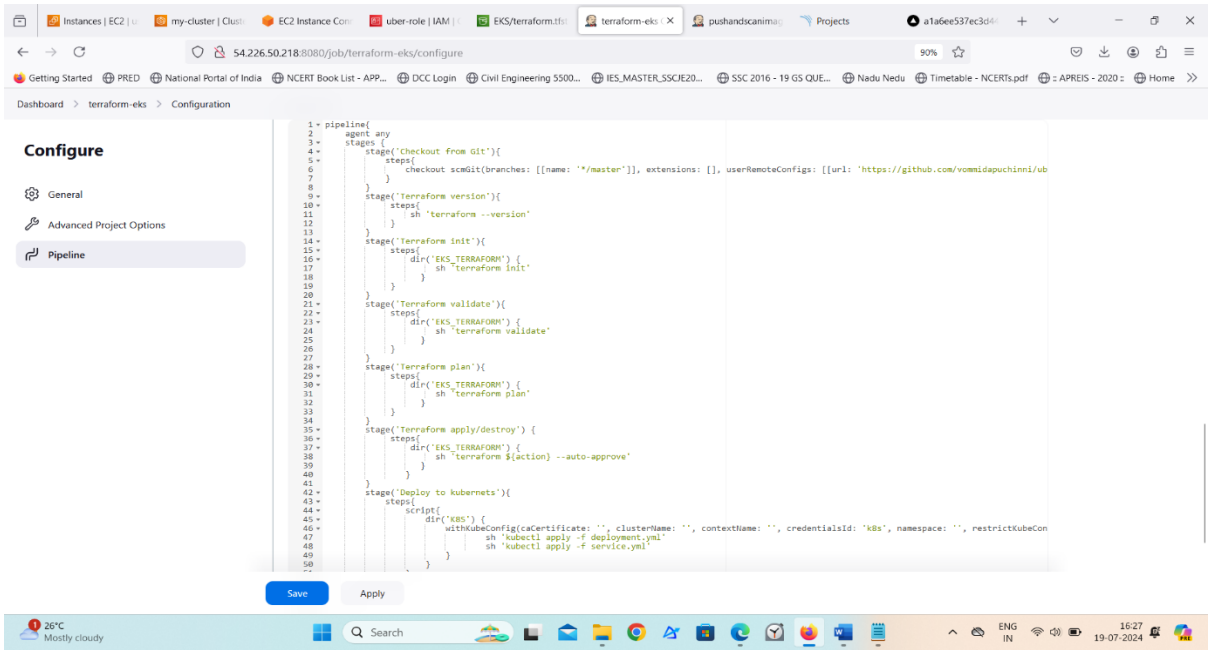


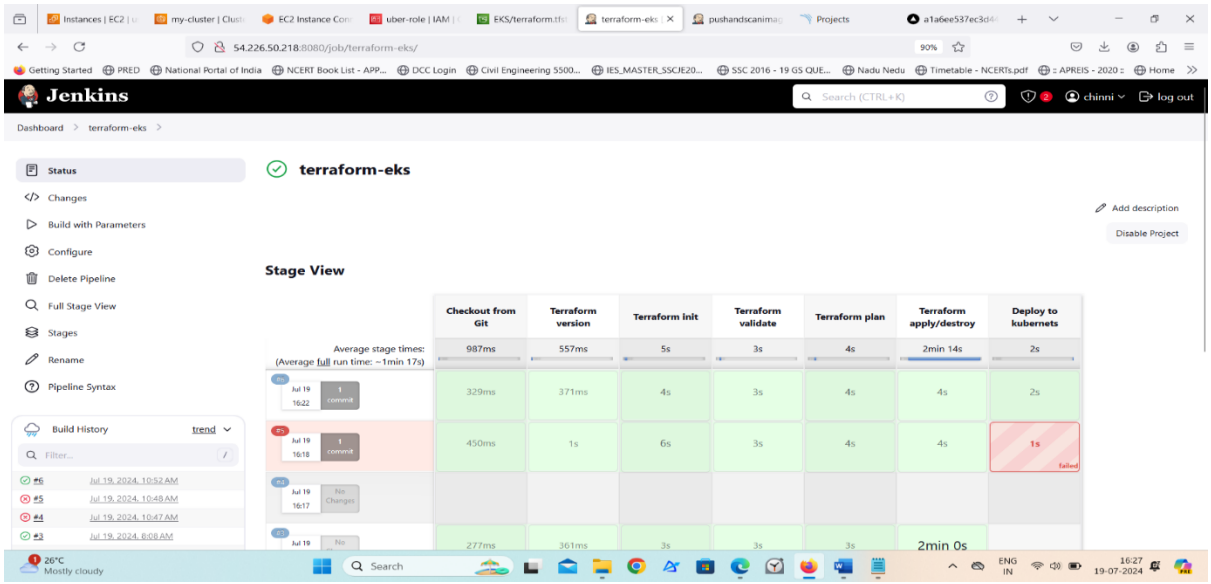
final step to deploy on the Kubernetes cluster

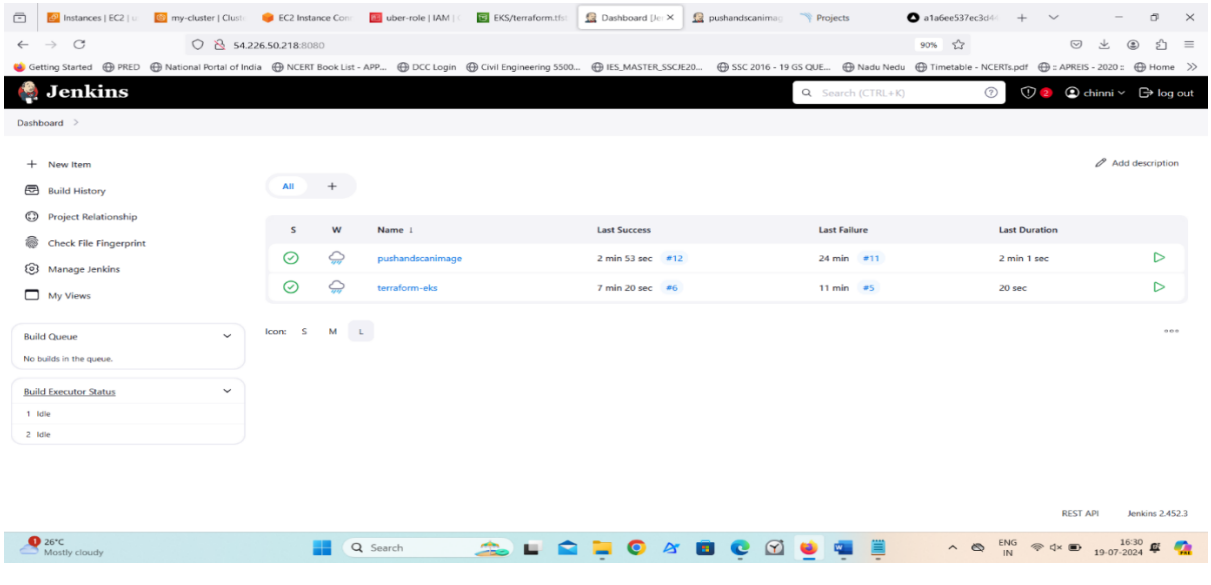
I deployed in terraform pipeline.

```
stage('Deploy to kubernets'){
        steps{
            script{
                dir('K8S') {
                    withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
                        sh 'kubectl apply -f deployment.yml'
                        sh 'kubectl apply -f service.yml'
                    }
                }
            }
        }
    }
```

After adding click on apply and then save it. Now click on build with parameters, choose apply as action click on build.


Configure
General
Advanced Project Options
Pipeline
pipeline{
agent any
stages {
stage('Checkout from Git'){
steps{
checkout scmGit(branches: [[name: '*/master']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/vommidapuchinni/ub
}
}
stage('Terraform version'){
steps{
sh 'terraform --version'
}
}
stage('Terraform init'){
steps{
dir('EKS_TERRAFORM') {
sh 'terraform init'
}
}
}
stage('Terraform validate'){
steps{
dir('EKS_TERRAFORM') {
sh 'terraform validate'
}
}
}
stage('Terraform plan'){
steps{
dir('EKS_TERRAFORM') {
sh 'terraform plan'
}
}
}
stage('Terraform apply/destroy') {
steps{
dir('EKS_TERRAFORM') {
sh 'terraform ${action} --auto-approve'
}
}
}
stage('Deploy to kubernets'){
steps{
script{
dir('K8S') {
withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeCon
sh 'kubectl apply -f deployment.yml'
sh 'kubectl apply -f service.yml'
}
}
Save
Apply

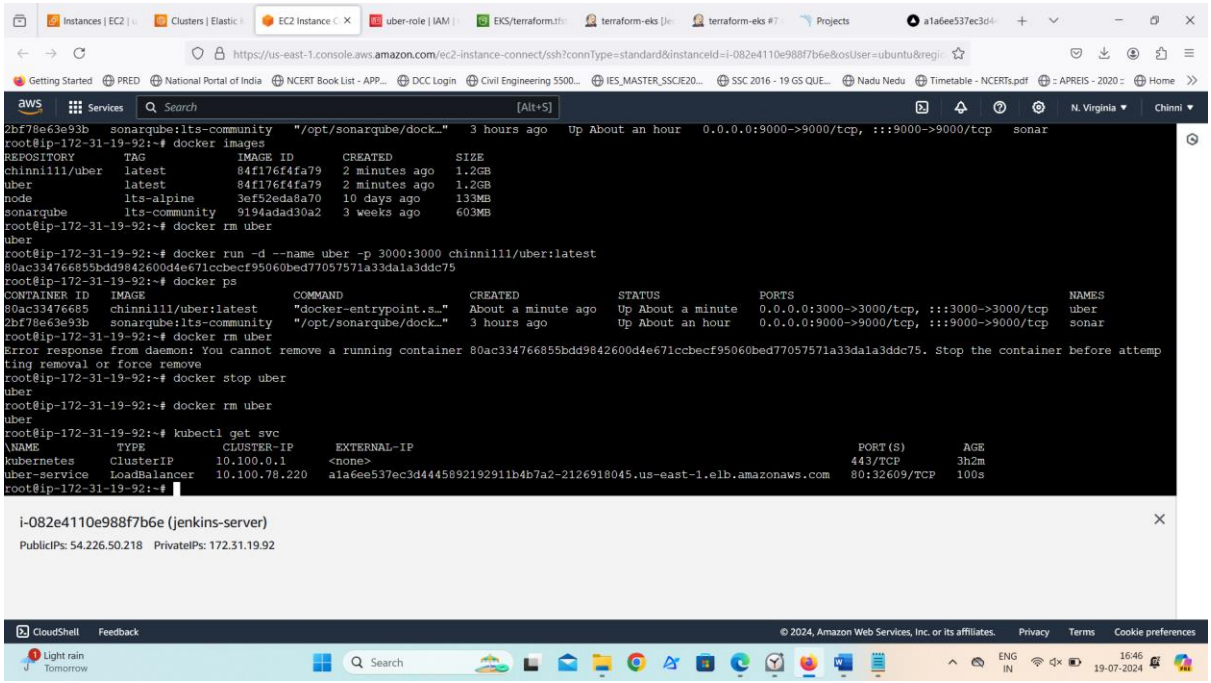

Jenkins
Dashboard
terraform-eks
Status
Changes
Build with Parameters
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Build History
trend
Add description
Disable Project
Stage View
Checkout from Git
Terraform version
Terraform init
Terraform validate
Terraform plan
Terraform apply/destroy
Deploy to kubernets
Average stage times:
(Average full run time: ~1min 17s)
987ms
557ms
5s
3s
4s
2min 14s
2s
329ms
371ms
4s
3s
4s
4s
2s
450ms
1s
6s
3s
4s
4s
1s
failed
277ms
361ms
3s
3s
3s
2min 0s


Jenkins
Dashboard
New Item
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Add description
All
S
W
Name
Last Success
Last Failure
Last Duration
pushandscanimage
2 min 53 sec #12
24 min #11
2 min 1 sec
terraform-eks
7 min 20 sec #6
11 min #5
20 sec
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
REST API
Jenkins 2.452.3

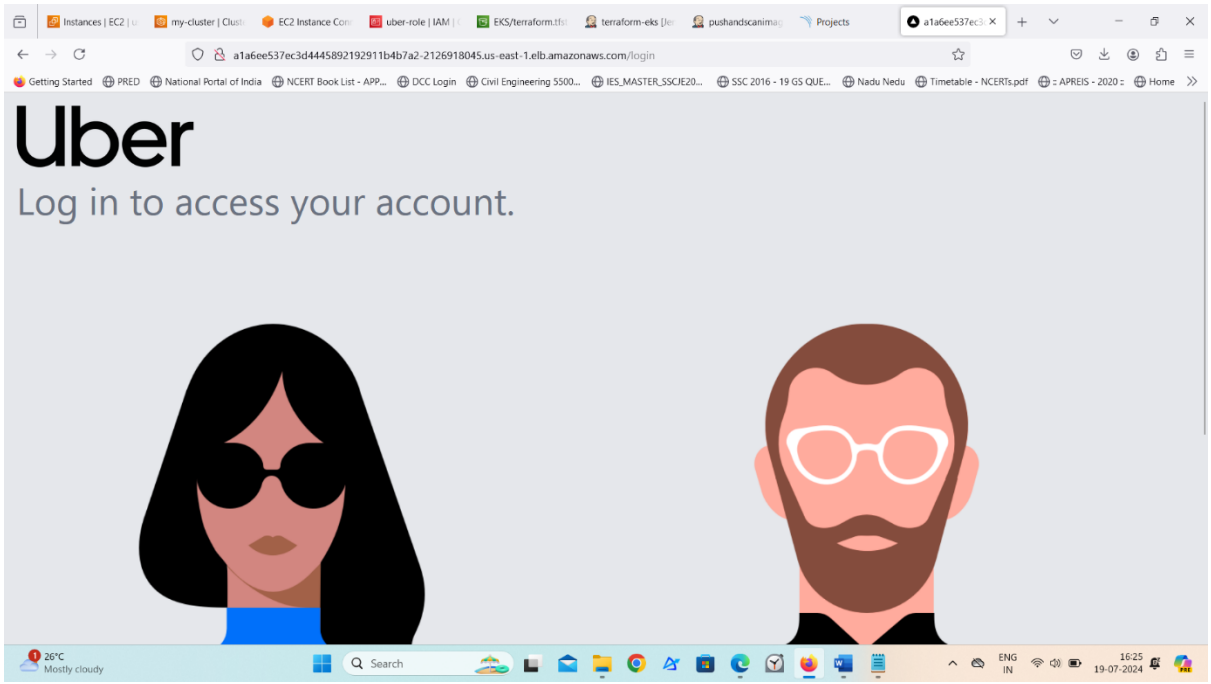
Step12: Accessing our application

Go to server give command as kubectl get svc.

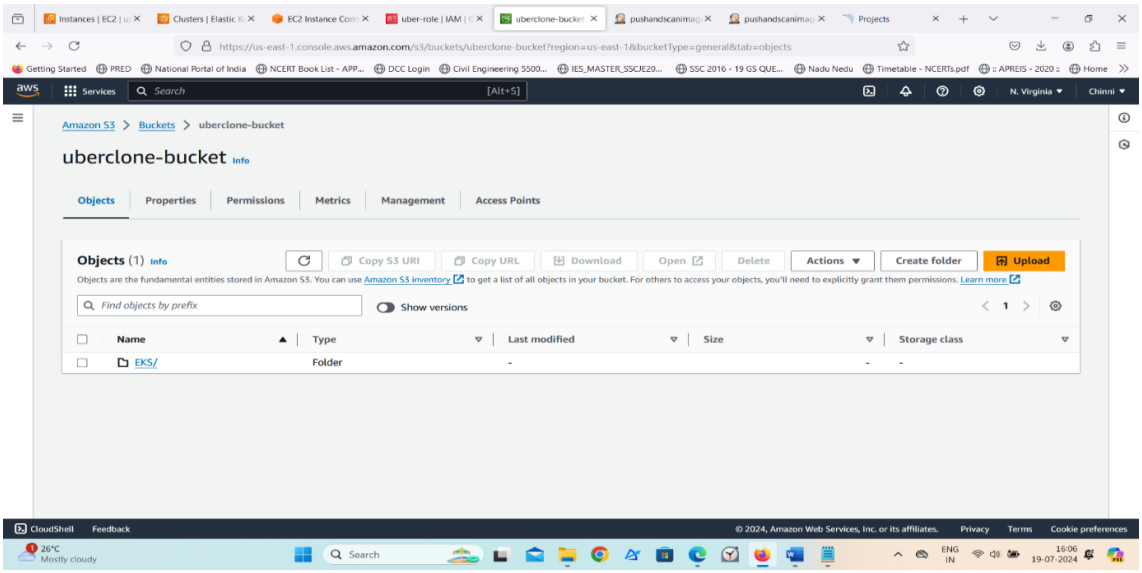
We see our services which are created.

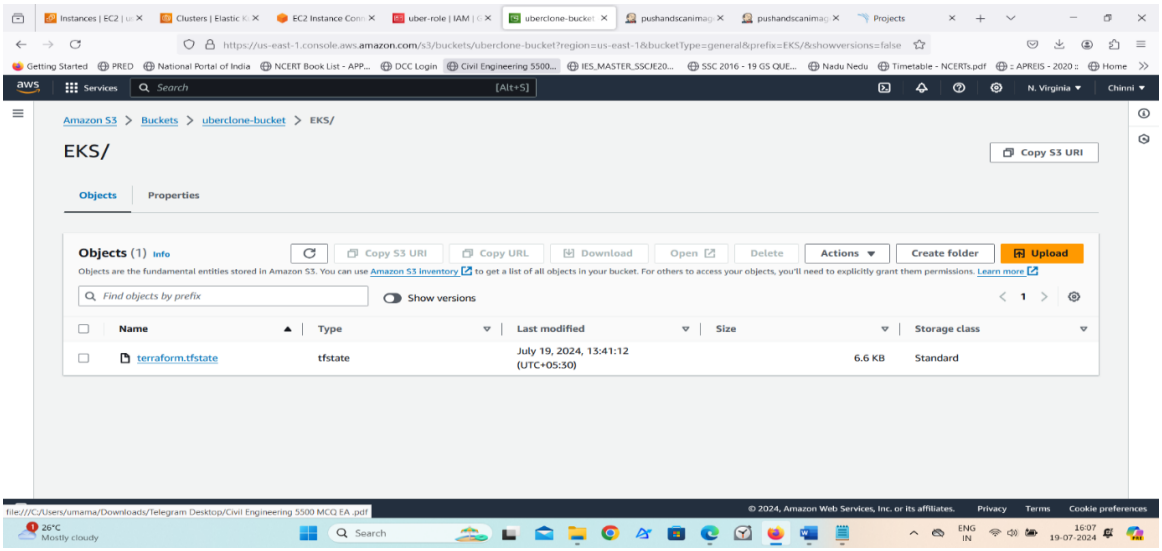


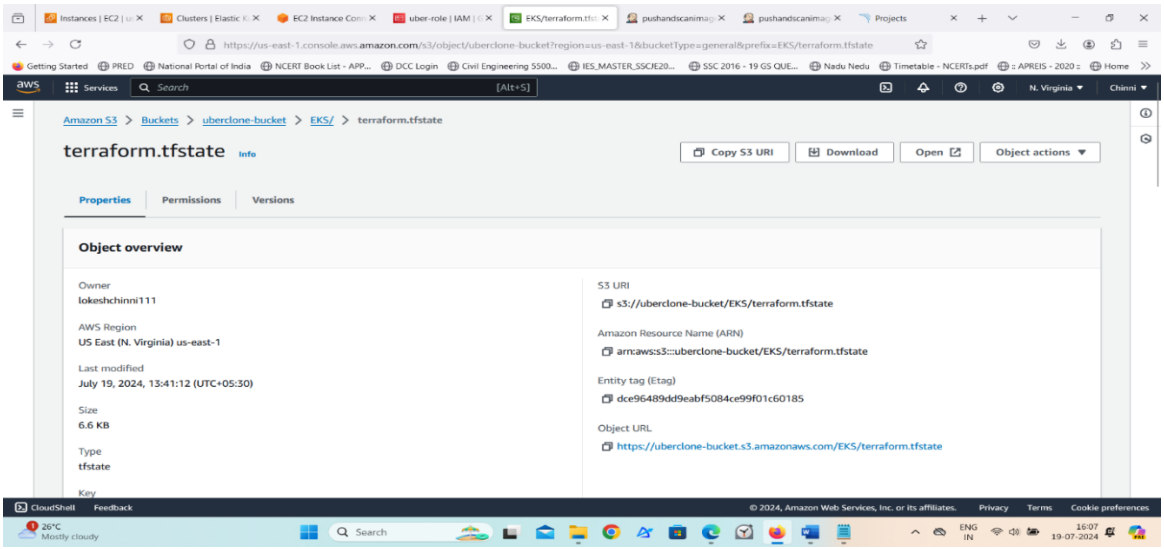
Copy load balancer external ip and paste on another tab we can see our webpage.



- Go to S3 bucket, click on created bucket.
- We see EKS folder in it. Click on it we see terraform statefile.





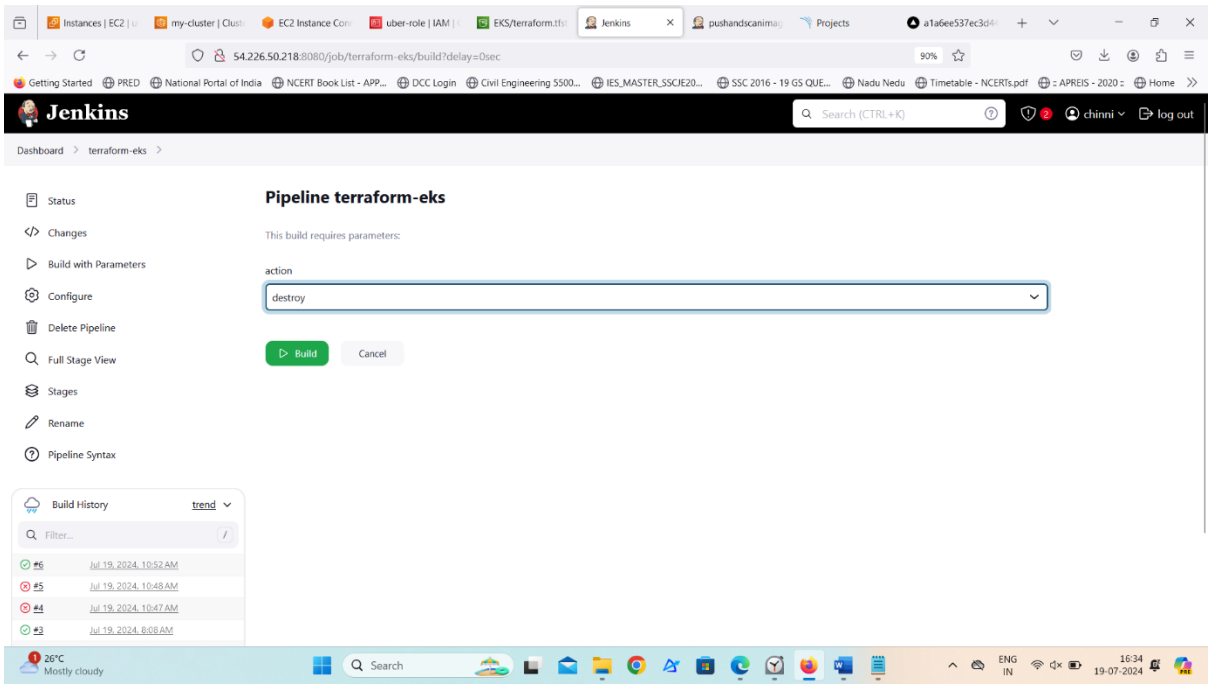


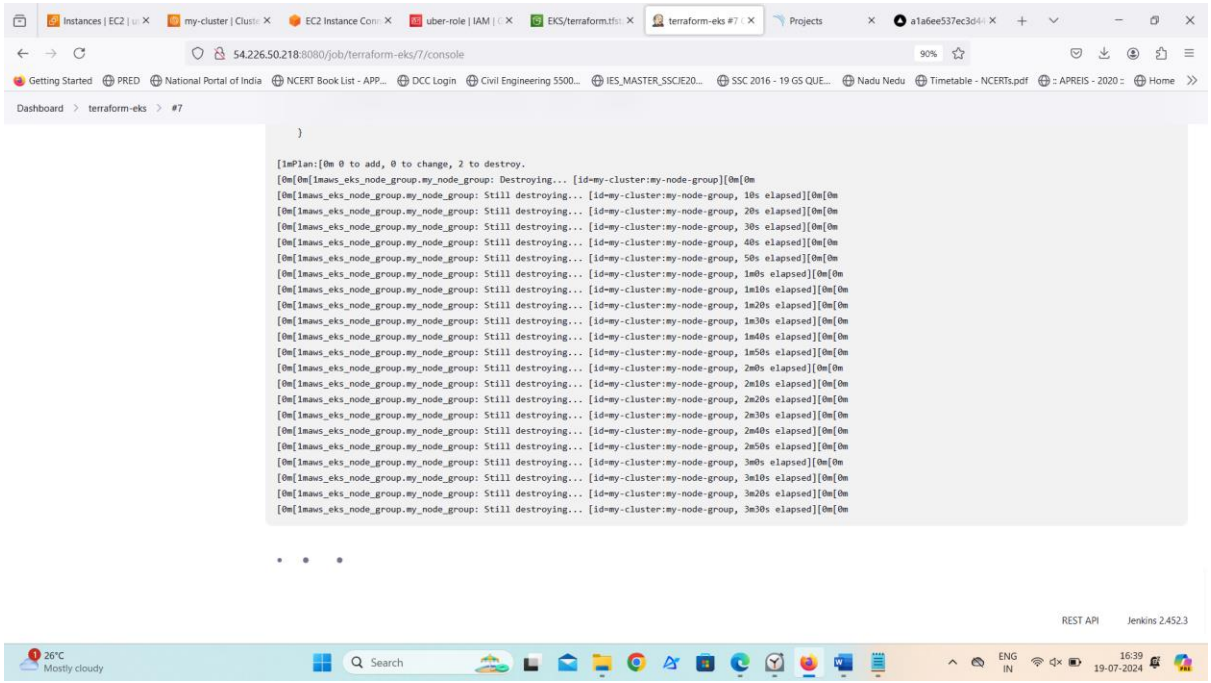
Step13: Destruction

Go to terraform-eks pipeline. Click on build with parameters.

Choose destroy as action click on build.

Until it destroys don’t delete anything.





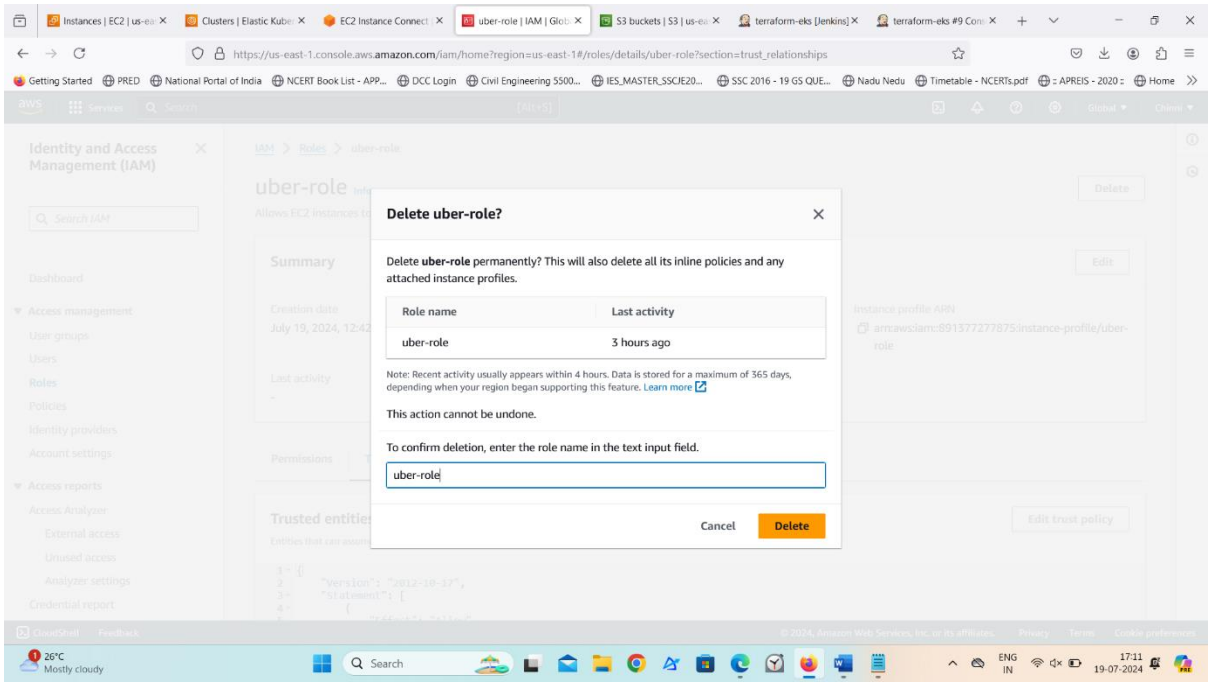
Delete two pipelines created and click on logout at right corner.

Logout from SonarQube server also.

Go to instance, click on instance state, click on terminate.

Go to S3 bucket and click on empty, after empty click on delete. Delete the bucket.

Also delete created IAM role.



Conclusion:

In conclusion, this documentation outlines a robust CI/CD pipeline tailored for our Uber clone project, aiming to optimize our development processes and streamline our delivery pipeline. By detailing each stage from code integration to automated testing and deployment, this documentation serves as a comprehensive guide for developers, testers, and operations teams. It emphasizes the importance of automation in ensuring code quality, enhancing collaboration, and achieving faster time-to-market. As we continue to iterate and improve our application, this CI/CD pipeline will play a pivotal role in maintaining reliability, scalability, and efficiency, ultimately contributing to the success of our Uber-like service.